



**Popular Integrated Circuit Program
Design and Examples**

流行集成电路程序 设计与实例

✦ 杨振江 冯 军 编著



西安电子科技大学出版社

<http://www.xduph.com>

XDUP 244600

封面设计 鄢 昭

中国大学网 华信教育资源网
www.hxedu.com.cn
北京 2008 年 8 月 1 日



- ▶ 介绍了日历器件、集成传感器、高精度A/D和D/A转换器、高性能存储器、无线通信模块、数字电位器、加密存储卡及电力电能器件等集成电路的性能与原理
- ▶ 内容简明扼要，所列器件突出使用技巧、应用实例和编程方法
- 日历时钟器件的使用与编程
- 集成传感器的使用与编程
- 高性能A/D、D/A转换器的使用与编程
- 高性能存储器的使用与编程
- 通信模块的使用与编程
- 数字电位器、电容器的使用与编程
- 电力电能器件、加密存储卡等的使用与编程

读者对象

从事智能仪器设计、数据采集、自动控制、数字通信和单片机应用的科技人员，以及工科院校通信、测控和自动化专业的师生

Popular Integrated Circuit Program Design and Examples

ISBN 978-7-5606-2154-8



9 787560 621548 >

定价：38.00元

流行集成电路程序设计与实例

杨振江 冯 军 编著

西安电子科技大学出版社

2009

内 容 简 介

本书从应用角度出发,精选了国内外最新流行的具有一定应用领域、特色鲜明、功能较强的新型集成电路,内容包括:日历时钟器件,集成传感器,A/D、D/A转换器,SPI、I²C器件,GSM/GPRS无线通信模块,GPS模块,数字电位器,高性能电力电能器件,接触式加密存储卡,非接触式通用读卡芯片等集成电路。对所选的每一种器件除阐述其硬件组成、基本功能、电路特点和引脚说明之外,更突出器件的使用实例和编程方法。所选器件均配有经过实践的源程序代码,有的例子直接来自于科学研究和生产实践,有些例子稍加修改就可用于解决工作中的实际问题。

本书对从事智能仪器设计、数据采集、自动控制、数字通信和计算机接口的科技人员和广大电子技术爱好者都具有很高的使用和参考价值,也可作为工科大专院校有关课程的教学参考书。

图书在版编目(CIP)数据

流行集成电路程序设计与实例 / 杨振江, 冯军编著. —西安: 西安电子科技大学出版社, 2009.2

ISBN 978-7-5606-2154-8

I. 流… II. ① 杨… ② 冯… III. 集成电路—电路设计 IV. TN402

中国版本图书馆 CIP 数据核字(2008)第 187124 号

策 划 云立实

责任编辑 许青青 云立实

出版发行 西安电子科技大学出版社(西安市太白南路2号)

电 话 (029)88242885 88201467 邮 编 710071

网 址 www.xduph.com 电子邮箱 xdupfxb001@163.com

经 销 新华书店

印刷单位 陕西华沐印刷科技有限责任公司

版 次 2009年2月第1版 2009年2月第1次印刷

开 本 787毫米×1092毫米 1/16 印 张 25.25

字 数 602千字

印 数 1~4000册

定 价 38.00元

ISBN 978-7-5606-2154-8/TN·0471

XDUP 2446001-1

如有印装问题可调换

本社图书封面为激光防伪覆膜, 谨防盗版。

前 言

我们正处在一个科学技术日新月异的时代。微电子技术的迅猛发展,使每一位从事或即将从事智能仪器和电子设备设计、开发及制造的工程技术人员都感到了前所未有的压力,都渴求能尽早掌握这些新技术、新原理和新器件的应用。过去人们在设计某些应用电路时,往往忽视了新的技术,而倾向采用过于陈旧的原理和器件。这些“散件”式组合设计在当时可称得上“最优”,但现在可能只需要一片集成电路就可取而代之,而且免去了装调、匹配等麻烦,降低了成本,提高了系统可靠性。因此,只有及时了解最新集成电路的功能及特点,才能用最少的器件设计出性能最优、功能最强的应用电路系统。

实际上,在掌握了必要的电路知识之后,每一位电路设计者都要遵循“精而巧”的指导思想。所谓“精”,就是尽可能地采用有“特点”的器件,如专用集成电路、多功能集成电路、混合集成电路等器件;所谓“巧”,就是所设计的电路要创新、新颖、实用。

另一方面,当今已出现了许多功能强大的集成电路芯片。这些芯片往往可以将一个具有完整功能的电路集成到一个芯片上,或将几个具有相关功能的电路甚至整套测控电路集成到一个芯片上。

大家知道,在设计一个硬件电路系统时,除了选择好的器件之外,程序设计也是重要的一环,它的质量直接影响到整个系统的性能。本书优选了较多 C51 源程序,用户可以借鉴所介绍的实例和某些电路的功能,方便地编写具体应用程序,解决编写程序中的困难,减少不必要的重复性工作。

编写本书的主要目的就是帮助那些从事智能仪器设计、数据采集板制作、自动化控制、数字通信和计算机接口编程的科技人员及其他电子技术爱好者很快掌握近年来国内外最新流行的常用与专用器件的主要特点、基本接法、典型应用和程序设计方法,使其更好地为科学研究和生产实践服务。

本书共 7 章,主要有日历时钟器件,集成传感器,高性能 A/D 转换器、D/A 转换器,高性能串行存储器,通信模块,数字电位器,智能存储射频卡控制器和电力电能器件等内容。

本书主要由杨振江和冯军编著,肖艺、王曙梅等同志参加了部分章节的编写。本书在编写过程中得到了云立实副编审和许青青编辑的大力支持与帮助,在此表示由衷的感谢。

由于编著者水平有限,书中不妥之处在所难免,恳请读者提出宝贵意见。

编著者
2008 年 10 月

目 录

第 1 章 日历时钟器件的使用与编程	1
1.1 RX8025A/AB 高性能实时时钟	1
1.1.1 硬件与功能描述	1
1.1.2 应用电路与编程	6
1.2 SD2098A 高精度实时时钟	11
1.2.1 硬件与功能描述	11
1.2.2 应用电路与编程	14
1.3 M41T0 低成本实时时钟	19
1.3.1 硬件与功能描述	19
1.3.2 应用电路与编程	22
1.4 PCF8563 实时时钟	25
1.4.1 硬件与功能描述	25
1.4.2 应用电路与编程	29
1.5 DS1302 低功耗自带 RAM 实时时钟	32
1.5.1 硬件与功能描述	32
1.5.2 应用电路与编程	37
第 2 章 集成传感器的使用与编程	44
2.1 DS18B20 数字温度传感器	44
2.1.1 硬件与功能描述	44
2.1.2 应用电路与编程	49
2.2 DS1631 温度传感器	53
2.2.1 硬件与功能描述	53
2.2.2 应用电路与编程	57
2.3 DS1722 温度传感器	61
2.3.1 硬件与功能描述	61
2.3.2 应用电路与编程	63
2.4 SHT1x/SHT7x 系列温湿度传感器	67
2.4.1 硬件与功能描述	67
2.4.2 应用电路与编程	71
2.5 MEMSIC 加速度传感器	76
2.5.1 硬件与功能描述	76
2.5.2 应用电路与编程	83
2.6 SCA100T 高精度倾角传感器	85

2.6.1	硬件与功能描述	85
2.6.2	应用电路与编程	88
第3章 高性能 A/D、D/A 转换器的使用与编程		92
3.1	AD5110 带有内部基准的 16 位 A/D 转换器	92
3.1.1	硬件与功能描述	92
3.1.2	应用电路与编程	96
3.2	MCP3221 低成本低功耗 12 位 A/D 转换器	102
3.2.1	硬件与功能描述	102
3.2.2	应用电路与编程	106
3.3	AD7705/06 用于低频测量的 16 位 A/D 转换器	108
3.3.1	硬件与功能描述	108
3.3.2	应用电路与编程	120
3.4	AD7714 高性能 24 位 A/D 转换器	123
3.4.1	硬件与功能描述	123
3.4.2	应用电路与编程	133
3.5	AD5320 低功耗满幅 12 位串行 D/A 转换器	136
3.5.1	硬件与功能描述	136
3.5.2	应用电路与编程	139
3.6	TLC5618 可编程双路 12 位串行 D/A 转换器	141
3.6.1	硬件与功能描述	141
3.6.2	应用电路与编程	145
3.7	AD9764 高精度高速 D/A 转换器	146
3.7.1	硬件与功能描述	146
3.7.2	应用电路与编程	150
第4章 高性能串行存储器的使用与编程		153
4.1	M25Pxx 大容量低成本 SPI 总线存储器	153
4.1.1	硬件与功能描述	153
4.1.2	应用电路与编程	159
4.2	EN25B32 高速大容量低成本 SPI 总线存储器	164
4.2.1	硬件与功能描述	164
4.2.2	应用电路与编程	168
4.3	FM25xx 系列高速高性能铁电 SPI 总线存储器	169
4.3.1	硬件与功能描述	169
4.3.2	应用电路与编程	176
4.4	FM24xx 系列高速高性能铁电 I ² C 总线存储器	181
4.4.1	硬件与功能描述	181
4.4.2	应用电路与编程	185
4.5	X5643/45 带有 CPU 监视的 E ² PROM 存储器	190
4.5.1	硬件与功能描述	190

4.5.2 应用电路与编程	196
4.6 X84161/641/129 低成本 E ² PROM 存储器	200
4.6.1 硬件与功能描述	200
4.6.2 应用电路与编程	204
4.7 93AA46/56/66 低压低功耗 E ² PROM 存储器	207
4.7.1 硬件与功能描述	207
4.7.2 应用电路与编程	212
第 5 章 通信模块的使用与编程	217
5.1 GTM900 GSM/GPRS 无线通信模块	217
5.1.1 硬件与功能描述	217
5.1.2 工作流程与系统组成	222
5.1.3 AT 命令集	222
5.1.4 应用电路与编程	230
5.2 JZ87x 系列无线数传通信模块	236
5.2.1 硬件与功能描述	236
5.2.2 通信协议的构建	238
5.2.3 应用电路与编程	238
5.3 nRF24L01 单片 2.4G 超低功耗数传器件	246
5.3.1 硬件与功能描述	246
5.3.2 器件的使用要点	258
5.3.3 应用电路与编程	259
5.4 新一代高性能 GPS 模块	263
5.4.1 总体描述	263
5.4.2 GPS 模块介绍	264
5.4.3 数据格式	268
5.4.4 应用电路与编程	273
第 6 章 数字电位器/电容器的使用与编程	278
6.1 X93154 低噪声低电压 32 抽头式数字电位器	278
6.1.1 硬件与功能描述	278
6.1.2 应用电路与编程	280
6.2 X9221A 双数控 64 抽头数字电位器	282
6.2.1 硬件与功能描述	282
6.2.2 应用电路与编程	286
6.3 X9318 数控 100 抽头数字电位器	291
6.3.1 硬件与功能描述	291
6.3.2 应用电路与编程	293
6.4 MCP41xxx 系列低功耗 256 抽头数字电位器	295
6.4.1 硬件与功能描述	295
6.4.2 应用电路与编程	298

6.5	X9111 低功耗 1024 抽头数字电位器	301
6.5.1	硬件与功能描述	301
6.5.2	应用电路与编程	304
6.6	CAT512x 系列 2 线接口数字电位器	311
6.6.1	硬件与功能描述	311
6.6.2	应用电路与编程	313
6.7	CAT5112 带缓冲滑动的 32 抽头数字电位器	315
6.7.1	硬件与功能描述	315
6.7.2	应用电路与编程	317
6.8	CAT5111 带缓冲滑动的 100 抽头数字电位器	318
6.8.1	硬件与功能描述	318
6.8.2	应用电路与编程	320
6.9	X90100 非易失性可编程电容器	322
6.9.1	硬件与功能描述	322
6.9.2	应用电路与编程	324
第 7 章	其他集成电路的使用与编程	326
7.1	CS5460A 高性能单片电力电能器件	326
7.1.1	硬件与功能描述	326
7.1.2	典型接法	335
7.1.3	应用电路与编程	336
7.2	MAX7219 单片数码管驱动器	343
7.2.1	硬件与功能描述	343
7.2.2	MAX7219 的使用与设置	345
7.2.2	应用电路与编程	348
7.3	SLE4432/42 接触式加密存储卡	350
7.3.1	硬件与功能描述	350
7.3.2	编程方法	355
7.4	MCP6S21/2/6/8 可编程增益模拟放大器	361
7.4.1	硬件与功能描述	361
7.4.2	应用说明	366
7.4.3	应用电路与编程	367
7.5	ISL88705/6/8/13/16 带有看门狗定时器的监控器件	369
7.5.1	硬件与功能描述	369
7.5.2	应用电路与编程	372
7.6	MF RC500 非接触式通用读卡芯片	374
7.6.1	硬件与功能描述	374
7.6.2	典型应用电路	382
7.6.3	编程方法	385

第1章 日历时钟器件的使用与编程

1.1 RX8025A/AB 高性能实时时钟

1.1.1 硬件与功能描述

RX8025A/AB 实时时钟器件内置高精度 32.768 kHz 的晶体振荡器和频率校准电路,能有效判断晶振是否停振,配有时钟精度可任意调整的控制电路,采用 I²C 总线接口,具有 6 种发生中断的设置,器件采用表贴封装,适合各种手机或便携式小型电子设备使用。

1. 主要性能特点

- (1) I²C 接口速率高达 400 kHz;
- (2) 年、月、日、时、分、秒和星期数据输出采用 BCD 码,可选择 12 或 24 时间制式;
- (3) 具有自动判断至 2099 年的闰年功能;
- (4) 内置根据环境温度调整计时的精度校准电路;
- (5) 中断请求可从 0.5 秒到 1 个月设置;
- (6) 有 2 个系统的闹钟功能;
- (7) 具有 32.768 kHz 时钟输出;
- (8) 内部具有 I²C 总线通信超时复位功能;
- (9) 能有效进行振荡停止检测;
- (10) 器件可在 1.7~5.5 V 的电源范围内工作;
- (11) 在 3 V 电源时,消耗电流仅为 0.48 μ A。

2. 内部框图与引脚说明

RX8025A/AB 实时时钟器件的内部结构框图如图 1-1 所示。

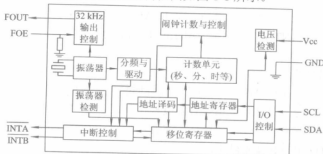


图 1-1 RX8025A/AB 的内部结构框图

RX8025A/AB 器件采用 SOP-14 和 SON-22 两种封装形式。其引脚排列如图 1-2 所示。

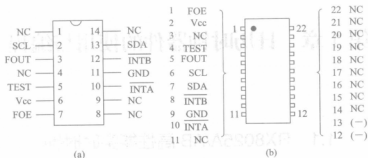


图 1-2 RX8025A/AB 的引脚封装图

(a) SOP-14 封装; (b) SON-22 封装

图 1-2 中:

- (1) SCL 是 I²C 接口时钟输入信号, 与 SDA 信号同步进行数据的传送。
- (2) SDA 是 I²C 接口数据输入/输出信号, 为开路输出时要接上拉电阻。
- (3) FOUT(FOE = 1 时)输出 32.768 kHz 信号, 为 CMOS 电平。当 FOE = 0 或 FOE 悬浮时, FOUT 输出低电平。
- (4) $\overline{\text{INTA}}$ 是 Alarm_D 闹钟中断漏极开路的输出信号。
- (5) $\overline{\text{INTB}}$ 是 Alarm_W 闹钟中断漏极开路的输出信号。
- (6) TEST 是测试信号, 正常时接 Vcc。
- (7) Vcc 是电源引脚。
- (8) GND 是接地引脚。
- (9) NC 是备用脚, 没有与内部 IC 连接, 一般接 GND。

3. 接口与特殊功能说明

1) RX8025A/AB 与 CPU 的接口

RX8025A/AB 采用标准的 I²C 接口, 即通过 SCL 和 SDA 两条信号线与 CPU 通信, 实现数据的读/写操作。SCL 的最大频率为 400 kHz, 与 I²C 总线高速模式相匹配。RX8025A/AB 规定时序从“开始”到“停止”必须在 0.5 秒内完成, 否则器件复位。

2) 闹钟功能

器件可在预先设定的时刻通过 Alarm_W 和 Alarm_D 对 CPU 发出中断警报。Alarm_D 只可设定、分报警从 $\overline{\text{INTA}}$ 引脚输出; Alarm_W 功能可设定星期、时、分报警从 $\overline{\text{INTB}}$ 引脚输出。

3) 计时精度调整功能

器件能够以 $\pm 3.05 \times 10^{-6}$ 的精度调高或调低计时精度, 通过使用这一功能可实现因季节因素对计时精度的影响。如果应用系统有温度检测功能, 则可根据温度的变化对计时精度进行修正, 从而使年内时间精度大大提高。注意, 只有计时精度可调整, 但不能从 FOUT 引脚输出 32.768 kHz 中反映调整结果。

4) 振荡停止检测与电源电压监视功能

振荡停止检测功能就是对振荡停止事件进行记忆的寄存器功能,通过这一功能可判断电源是否变为0V或后备电源是否降低,从而判断此时计时数据是否有效。

电源电压检测是比较并判定电源电压是否低于设定值的检测,方法是由寄存器设定2.1V和1.3V两种电压中的一种,每秒抽样进行电源电压监视。

5) 固定周期发生中断功能

除闹钟功能外,可从INTA引脚输出固定周期的中断,即可从2Hz(0.5秒)、1Hz(1秒)、1/60Hz(每分)、1/3600Hz(每时)、每月(各月的1天)共5种频率中选择输出。

固定周期中断的输出波形有两种,即脉冲波形(2Hz、1Hz)和CPU中断的电平波形(每秒、每分、每时、每月)。

4. 寄存器说明

RX8025A/AB寄存器的地址和含义如表1-1所示。其中:PON位是电源复位标志,刚接通电源时或电源电压下降复位后,PON位设置为1,且除PON位及XST位以外的Control 1、Control 2的各位重设为0,这时其他寄存器不稳定,因而此时不要设定日期和时间,以免出现计时错误;TEST位是器件商测试专用位,请务必设定为0;地址D(保留寄存器)为厂商专用位,请不要进行读/写操作;“0”标记位请务必设定为0后再使用;“○”标记位不能写入,读取为0;“△”标记位为可“R/W”任意数据的RAM位,但PON位为1时被清除为0。

表 1-1 RX8025A/AB 寄存器的地址和含义

地址	功能	位7	位6	位5	位4	位3	位2	位1	位0
0	秒	○	S40	S20	S10	S8	S4	S2	S1
1	分	○	M40	M20	M10	M8	M4	M2	M1
2	时	○	○	H20	H10	H8	H4	H2	H1
3	星期	○	○	○	○	○	W4	W2	W1
4	日	○	○	D20	D10	D8	D4	D2	D1
5	月	0	○	○	MO10	MO8	MO4	MO2	MO1
6	年	Y80	Y40	Y20	Y10	Y8	Y4	Y2	Y1
7	数字偏移量	0	F6	F5	F4	F3	F2	F1	F0
8	闹钟 W:分	○	WM40	WM20	WM10	WM8	WM4	WM2	WM1
9	闹钟 W:时	○	○	WH20	WH10	WH8	WH4	WH2	WH1
A	闹钟 W:星期	○	WW6	WW5	WW4	WW3	WW2	WW1	WW0
B	闹钟 D:分	○	DM40	DM20	DM10	DM8	DM4	DM2	DM1
C	闹钟 D:时	○	○	DH20	DH10	DH8	DH4	DH2	DH1
D	保留	保留							
E	控制(Control 1)	WALE	DALE	$\overline{12/24}$	△	TEST	CT2	CT1	CT0
F	控制(Control 2)	VDSL	VDET	\overline{XST}	PON	△	CTFG	WAFG	DAFG

1) 控制寄存器 1(Control 1)

控制寄存器1在上电后(初始接通电源后)默认值为0。其中:

(1) WALE 位=0 时, 闹钟 W(Alarm_W)比较动作无效; WALE 位=1 时, 比较动作有效(报警时, $\overline{\text{INTB}}=0$)。

(2) DALE 位=0 时, 闹钟 D(Alarm_D)比较动作无效; DALE 位=1 时, 比较动作有效(报警时, $\overline{\text{INTA}}=0$)。

(3) $\overline{\text{I2/24}}$ 位=0 时, 为 12 小时制; $\overline{\text{I2/24}}$ 位=1 时, 为 24 小时制。

(4) CT2、CT1、CT0 位是对 INTA 脚中断输出状态的设定, 为 000 时, $\overline{\text{INTA}}$ 为高阻; 为 001 时, $\overline{\text{INTA}}$ 为 0; 为 010 时, $\overline{\text{INTA}}$ 为 2 Hz 波形输出; 为 011 时, $\overline{\text{INTA}}$ 为 1 Hz 波形输出; 为 100 时, $\overline{\text{INTA}}$ 为 1 秒 1 次(与秒递增计数同步); 为 101 时, $\overline{\text{INTA}}$ 为 1 分 1 次(每分 00 秒); 为 110 时, $\overline{\text{INTA}}$ 为 1 小时 1 次(每时 00 分 00 秒); 为 111 时, $\overline{\text{INTA}}$ 为 1 月 1 次(每月 1 日上午 00 时 00 分)。

2) 控制寄存器 2(Control 2)

控制寄存器 2 在上电后(初始接通电源后)默认值为 0x10000B。其中:

(1) VDSL 位=0 时, 将电源降低检测值设为 2.1 V; VDSL 位=1 时, 将电源降低检测值设为 1.3 V。

(2) VDET 位是低电压检测功能的检测结果位, 将 VDET 位设为 0(禁止设为 1)时, 表示重新开始低电压检测。当读到 VDET 位=0 时, 表示电压正常; 当读到 VDET 位=1 时, 表示检测到电压降低。

(3) $\overline{\text{XST}}$ 位是振荡停止检测结果位, 将 $\overline{\text{XST}}$ 位设为 0 时禁止设定; 将 $\overline{\text{XST}}$ 位设为 1 时允许设定。当读到 $\overline{\text{XST}}$ 位=0 时, 表示振荡器停止; 当读到 $\overline{\text{XST}}$ 位=1 时, 表示振荡器正常。

(4) PON 位是电源复位的检测结果位, 将 PON 位设为 0, 表示允许检测; 将 PON 位设为 1, 表示禁止检测。当读到 PON 位=0 时, 表示电源正常; 当读到 PON 位=1 时, 表示有过电源复位发生。

(5) CTFG 位是 $\overline{\text{INTA}}$ 引脚固定周期中断输出状态控制位, 将 CTFG 位设为 0 表示关掉 $\overline{\text{INTA}}$ 引脚固定输出中断(禁止设为 1)。当读出 CTFG 位=0 时, 表示关掉固定输出功能; 当读出 CTFG 位=1 时, 表示固定输出功能有效。

(6) WAFG 位是 $\overline{\text{INTB}}$ 引脚中断输出状态控制位, 将 WAFG 位设为 0 表示关掉 $\overline{\text{INTB}}$ 引脚中断(禁止设为 1)。当读出 WAFG 位=0 时, 表示设定时刻和现行时刻不一致; 当读出 WAFG 位=1 时, 表示设定时刻和现行时刻一致。

(7) DAFG 位是 $\overline{\text{INTA}}$ 引脚中断输出状态控制位, 将 DAFG 位设为 0 表示关掉 $\overline{\text{INTA}}$ 引脚中断(禁止设为 1)。当读出 DAFG 位=0 时, 表示设定时刻和现行时刻不一致; 当读出 DAFG 位=1 时, 表示设定时刻和现行时刻一致。

3) 计时计数器

计时计数器包括“秒”、“分”、“时”、“星期”、“日”、“月”和“年”。数据为 BCD 码。

对于“时”计数的“位 5, H20”位, 在 24 制式时, 为 BCD 码值; 在 12 制式时, “位 5, H20”=0 表示 AM(上午), “位 5, H20”=1 表示 PM(下午)。

4) 闹钟计数器

闹钟计数器包括“闹钟 W”和“闹钟 D”。数据为 BCD 码。其中, “时”闹钟的位 5(WH20 和 DH20), 当地时间制式为 24 小时制时, 为“时”的 BCD 码数; 当地时间制式为 12 小时制时,

该“位”为AM/PM。

5) 计时精度调整寄存器(数字偏移量)

计时精度调整寄存器上电后,默认值为0。如果不调整,则必须将该寄存器设为0。

(1) 计时精度调整寄存器的设置。通过改变F6~F0的7位二进制数据,可以调整计时的快慢。调整范围为 $-189.10 \times 10^{-6} \sim +189.10 \times 10^{-6}$;调整分辨率为 $\pm 3.05 \times 10^{-6}$;内部的调整实施时间为20秒一次。调整偏移量如表1-2所示。

表 1-2 计时精度调整偏移量

调整量 ($\times 10^{-6}$)	调整数据 十进制/十六进制	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
		0	F6	F5	F4	F3	F2	F1	F0
-189.10	+63/3F	0	0	1	1	1	1	1	1
-186.05	+62/3E	0	0	1	1	1	1	1	0
-183.00	+61/3D	0	0	1	1	1	1	0	1
...							
-9.15	+4/04	0	0	0	0	0	1	0	0
-6.10	+3/03	0	0	0	0	0	0	1	1
-3.05	+2/02	0	0	0	0	0	0	1	0
OFF	+1/01	0	0	0	0	0	0	0	1
OFF	0/00	0	0	0	0	0	0	0	0
+3.05	-1/7F	0	1	1	1	1	1	1	1
+6.10	-2/7E	0	1	1	1	1	1	1	0
+9.15	-3/7D	0	1	1	1	1	1	0	1
...							
+183.00	-60/44	0	1	0	0	0	1	0	0
+186.05	-61/43	0	1	0	0	0	0	1	1
+189.10	-62/42	0	1	0	0	0	0	1	0
OFF	-63/41	0	1	0	0	0	0	0	1
OFF	-64/40	0	1	0	0	0	0	0	0

(2) 计时精度调整举例。

【例 1】当 FOUT 计时器输出为 32 767.7 Hz 时,调整为 32 768 Hz 的计数。

① 偏移量为

$$(32767.7 - 32768) \div 32768 \approx -9.16 \times 10^{-6}$$

② 对目前偏移量的最佳调整数据为

调整数据 = 偏移量 \div 调整分辨率 = $-9.16 \div 3.05 \approx -3$ (把小数点四舍五入以后)

③ 计算出设定调整的十六进制数据。由 7 位二进制数计算出被设定的调整数据,应从 128(80H)中减去要调整的数据,即设定调整数据 = $128 - 3 = 125$,或 $80H - 03H = 7DH$,也就是将 7DH 写入“数字偏移量”寄存器中即可得到精度补偿。

【例2】当 FOUT 计时器输出为 32 768.3 Hz 时, 调整为 32 768 Hz 的计数。

① 偏移量为

$$(32768.3 - 32768) \div 32768 \approx +9.16 \times 10^{-6}$$

② 对目前偏移量的最佳调整数据为

调整数据 = 偏移量 ÷ 调整分辨率 + 1 = $+9.16 \div 3.05 + 1$ (因标准为 1H, 故加 1)

$\approx +4$ (小数点四舍五入以后)

③ 计算出设定调整的十六进制数据。设定调整数据 = 04H, 即将 04H 写入“数字偏移量”寄存器中即可得到精度补偿。

5. 读/写操作说明

1) 写操作

“写操作”的顺序是: 开始条件、写入“64H”、“寄存器地址(高 4 位) + 0H(低 4 位)”、“被写入的数据”、停止条件。

2) 读操作

“读操作”有两种方法: 第一种为标准读法, 即: 开始条件、写入“64H”、“寄存器地址(高 4 位) + 0H(低 4 位)”、开始条件、写入“65H”、“数据 1”、“数据 2”、……、停止条件; 第二种为缩短顺序读法, 即: 开始条件、写入“64H”、“寄存器地址(高 4 位) + 4H(低 4 位)”、“数据 1”、“数据 2”、……、停止条件。

1.1.2 应用电路与编程

RX8025A 高性能实时时钟与 STC89C52 单片机的接口电路如图 1-3 所示。这是一个标准的 I²C 总线接口。总线时序分为开始条件、结束条件、数据传送、确认。总线从地址写是 64H, 读是 65H。时序可参考后面的图 1-11~图 1-13。

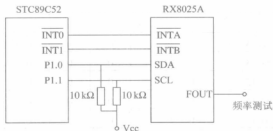


图 1-3 RX8025A 与单片机的接口电路

用 C51 程序编写的“写”、“读”等实用函数如下:

```
void Delay(void)                /*延时函数*/
{
    unsigned char i;
    for(i=0; i<2; i++);        /*延时*/
}
```

/*功能: 向 RX8025A 的寄存器里写入一个字节(包括总线初始化、开始与终止条件)。

参数: reg 表示目的寄存器地址 0x00~0x0F;

```

    dat 表示要写入的字节    */
void Write8025_i2c(unsigned char reg,unsigned char datai2c)
{
    unsigned char dat,b,i,ack; /*分别用于形参传递、8位字长计数、延时等*/
    SDA=1; SCL=1;             /*初始化*/
    Delay(); SDA=0;            /*启动*/
    Delay(); SCL=0;             /*延时*/
    dat=0x64;                   /*发送写地址*/
    for(b=0; b<8; b++)
    {
        if(dat&0x80)            /*高位在前*/
            SDA=1;
        else
            SDA=0;              /*先准备好数据*/
        Delay(); SCL=1;         /*再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
        Delay(); SCL=0;
        Delay(); dat<<=1;
    }
    SDA=1; Delay();            /*立即释放数据线*/
    SCL=1;
    if(SDA==0)                 /*接收 ACK*/
        ack=0;
    else
        ack=1;
    SCL=0;
    /*发送寄存器地址。高 4 位为目的寄存器地址；低 4 位为 0000，为固定的写模式*/
    dat=(reg<<4);
    for(b=0; b<8; b++)
    {
        if(dat&0x80)            /*高位在前*/
            SDA=1;
        else
            SDA=0;              /*先准备好数据*/
        Delay(); SCL=1;         /*再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
        Delay(); SCL=0;
        Delay(); dat<<=1;
    }
    SDA=1; Delay();            /*立即释放数据线*/
    SCL=1;

```

```

    if(SDA==0)                /*接收 ACK*/
        ack=0;
    else
        ack=1;
    SCL=0;
    dat=datai2c;               /*发送数据*/
    for(b=0; b<8; b++)
    {
        if(dat&0x80)          /*高位在前*/
            SDA=1;
        else
            SDA=0;             /*先准备好数据*/
        Delay(); SCL=1;        /*再在时钟线上发出一个正脉冲, 最后 SCL 被置 0*/
        Delay(); SCL=0;
        Delay(); dat<<=1;
    }
    SDA=1; Delay();           /*立即释放数据线*/
    SCL=1;
    if(SDA==0)                /*接收 ACK*/
        ack=0;
    else
        ack=1;
    SCL=0; SDA=0;             /*停止条件*/
    Delay(); SCL=1;
    Delay(); SDA=1;
}

```

/*功能: 从 RX8025A 的寄存器中读出一个字节并作为返回值(包括开始与终止条件)。

参数: reg 表示目的寄存器地址 0x00~0x0F(RX8025A 内部的寄存器 s、m、h、...)*/

unsigned char Read8025_i2c(unsigned char reg)

```

{
    unsigned char datai2c;     /*读回的数据*/
    unsigned char dat,b,i,ack; /*分别用于形参传递、8 位字长计数、延时、取应答位*/
    SDA=1; SCL=1;             /*初始化*/
    Delay(); SDA=0;           /*启动*/
    Delay(); SCL=0;
    dat=0x64;                  /*发送写地址*/
    for(b=0; b<8; b++)
    {
        if(dat&0x80)          /*高位在前*/
            SDA=1;

```

```

else
    SDA=0;          /*先准备好数据*/
    Delay(); SCL=1;  /*再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
    Delay(); SCL=0;
    Delay(); dat<=1;
}
SDA=1;    Delay(); /*立即释放数据线*/
SCL=1;
if(SDA==0) /*接收 ACK*/
    ack=0;
else
    ack=1;
SCL=0;
/*发送目的寄存器地址。高 4 位为目的寄存器地址；低 4 位为 0100，即缩短读模式 2*/
dat=((reg<<4)|0x04);
for(b=0; b<8; b++)
{
    if(dat&0x80) /*高位在前*/
        SDA=1;
    else
        SDA=0; /*先准备好数据*/
    Delay(); SCL=1; /*再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
    Delay(); SCL=0;
    Delay(); dat<=1;
}
SDA=1; Delay(); /*立即释放数据线*/
SCL=1;
if(SDA==0) /*接收 ACK*/
    ack=0;
else
    ack=1;
SCL=0;
for(b=0; b<8; b++) /*读数据*/
{
    datai2c<=1;
    Delay(); /*让从器件准备数据*/
    SCL=1; /*在 SCL 为高电平的情况下读 SDA*/
    if(SDA==0)
        datai2c|=0xfe; /*datai2c 末位清 0*/
    else

```



```

        datai2c=0x01;          /*datai2c 末位置 1*/
        SCL=0;                 /*时钟线置低*/
    }
    SCL=0; Delay();            /*发送 Nack, 通知从机停止发送*/
    SDA=1; Delay();            /*准备好数据*/
    SCL=1; Delay();
    SCL=0; Delay();
    SDA=0; Delay();            /*停止条件*/
    SCL=1; Delay();
    SDA=1;
    return(datai2c);
}

void Write_Rx8025_inte(void)    /*初始化函数*/
{
    Write8025_i2c(0x07,0x00);   /*不调整精度*/
    Write8025_i2c(0x08,0x00);   /*不报警*/
    Write8025_i2c(0x09,0x00);   /*不报警*/
    Write8025_i2c(0x0a,0x00);   /*不报警*/
    Write8025_i2c(0x0b,0x00);   /*不报警*/
    Write8025_i2c(0x0c,0x00);   /*不报警*/
    Write8025_i2c(0x0e,0x23);   /*24 小时制*/
    Write8025_i2c(0x0f,0xc4);
}

void Wdata_time8025(y,mo,d,h,m,s) /*写入时间与日历函数*/
    unsigned char y,mo,d,h,m,s;
{
    Write8025_i2c(0x00,s);      /*秒*/
    Write8025_i2c(0x01,m);      /*分*/
    Write8025_i2c(0x02,h);      /*时*/
    Write8025_i2c(0x04,d);      /*日*/
    Write8025_i2c(0x05,mo);     /*月*/
    Write8025_i2c(0x06,y);      /*年*/
}

/*****读时钟*****/
void Read_date_time(y)          /*读年、月、日、时、分、秒函数*/
    unsigned char y[];
{
    y[0]=Read8025_i2c(0x06);     /*年*/
    y[1]=Read8025_i2c(0x05);     /*月*/
    y[2]=Read8025_i2c(0x04);     /*日*/

```

```

y[3]=Read8025_i2c(0x02);          /*时*/
y[4]=Read8025_i2c(0x01);          /*分*/
y[5]=Read8025_i2c(0x00);          /*秒*/
}

```

1.2 SD2098A 高精度实时时钟

1.2.1 硬件与功能描述

SD2098A 是一种具有内置 2 线式串行接口的实时时钟芯片。该芯片具有时钟精度数字调整功能,可在很宽的范围内校正时钟的偏差(分辨率为 3×10^{-6}),通过外置的温度传感器可设定适应温度变化的调整值,实现宽温度范围内高精度的计时功能。

1. 主要性能特点

- (1) 低功耗,典型值为 $0.5 \mu\text{A}$ ($V_{\text{CC}}=3.0 \text{ V}$);
- (2) 工作电压为 $1.8 \sim 5.5 \text{ V}$,工作温度为 $-40 \sim +85^\circ\text{C}$;
- (3) 具有年、月、日、星期、时、分、秒的 BCD 码输入/输出形式,并可通过独立的地址访问各时间寄存器;
- (4) 自动日历到 2099 年(包括闰年自动换算功能);
- (5) 可设定并自动重置两路定时闹钟功能(时间范围在 1 周内);
- (6) 周期性中断脉冲输出: 2 Hz 、 1 Hz 、每分钟、每小时、每个月信号,输出可选择不同波形的中断脉冲;
- (7) 可控的 $32\,768 \text{ Hz}$ 方波信号输出;
- (8) 内置时钟精度数字调整功能;
- (9) 30 秒时间调整功能;
- (10) 内置晶振停振检测功能,保证时钟的可靠性和有效性;
- (11) 内置总线 1 秒自动释放功能,保证时钟数据线的有效性;
- (12) 内置稳压电源,内部计时电压可低至 1.2 V 。

2. 内部结构与引脚说明

SD2098A 实时时钟器件的内部结构与引脚排列如图 1-4 所示。

SD2098A 采用 SOP-8 脚封装。其中:

- (1) INTRB 是报警中断 B 开路输出脚,根据控制寄存器 1 与 2 来设置其工作模式,当定时时间到达时输出低电平或时钟信号。它可通过重写状态寄存器来禁止。
- (2) SCL 是 I^2C 接口时钟输入信号,与 SDA 信号同步进行数据传送。
- (3) SDA 是 I^2C 接口数据输入/输出信号,为开路输出,要接上拉电阻。
- (4) GND 是参考地。
- (5) INTRA 是报警中断 A 开路输出脚,根据控制寄存器 1 与 2 来设置其工作模式,当定时时间到达时输出低电平或时钟信号。它可通过重写状态寄存器来禁止。
- (6) Xo、Xi 是晶振的输出脚与输入脚。

(7) V_{CC} 是电源脚。

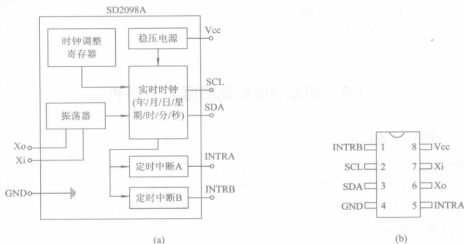


图 1-4 SD2098A 器件的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

3. 内部寄存器说明

SD2098A 内部寄存器的地址与功能如表 1-3 所示。表中地址 0H~6H 和地址 8H~DH 均为 BCD 码。

表 1-3 SD2098A 内部寄存器的地址与功能

地 址	功 能	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
0H	秒	—	S40	S20	S10	S8	S4	S2	S1
1H	分	—	M40	M20	M10	M8	M4	M2	M1
2H	时	—	—	H20	H10	H8	H4	H2	H1
3H	星期	—	—	—	—	—	W4	W2	W1
4H	天	—	—	D20	D10	D8	D4	D2	D1
5H	月	—	—	—	MO10	MO8	MO4	MO2	MO1
6H	年	Y80	Y40	Y20	Y10	Y8	Y4	Y2	Y1
7H	时间调整	XSL	F6	F5	F4	F3	F2	F1	F0
8H	分定时 A	—	AM40	AM20	AM10	AM8	AM4	AM2	AM1
9H	时定时 A	—	—	AH20	AH10	AH8	AH4	AH2	AH1
AH	星期定时 A	—	AW6	AW5	AW4	AW3	AW2	AW1	AW0
BH	分定时 B	—	BM40	BM20	BM10	BM8	BM4	BM2	BM1
CH	时定时 B	—	—	BH20	BH10	BH8	BH4	BH2	BH1
DH	星期定时 B	—	BW6	BW5	BW4	BW3	BW2	BW1	BW0
EH	控制 1	AALE	BALE	SL2	SL1	TEST	CT2	CT1	CT0
FH	控制 2	—	—	$\overline{I2_24}$	ADJ	\overline{CLEN}	CTFG	AAFG	BAFG

其中:

(1) 在 12 小时制时, “时位 5, H20/AH20/BH20” 的值是 PM/AM, 当 “时位 5, H20/AH20/BH20” = 0 时, 表示 AM, 当 “时位 5, H20/AH20/BH20” = 1 时, 表示 PM。

(2) 时间调整寄存器中的 XSL 位必须设为 0, F6~F0 是时间调整偏差数据。时间调整是根据预先设置的数据在秒计数为 00、20、40 时刻, 改变 1 秒时钟内计数的个数。通常每 32 768 个脉冲为 1 秒(对寄存器预设初值, 才能激活整个调整电路)。

当 F6 为 0 时, 产生 1 秒的寄存器计数脉冲, 将增加为: $32\,768 + ((F5, F4, F3, F2, F1, F0) - 1) \times 2$ 的值;

当 F6 为 1 时, 产生 1 秒的寄存器计数脉冲, 将减少为: $32\,768 - ((F5, F4, F3, F2, F1, F0) + 1) \times 2$ 的值(F5 是 F5 的反码, 其他同理);

当(F6, F5, F4, F3, F2, F1, F0)预设为(*, 0, 0, 0, 0, 0, *)时, 产生 1 秒的寄存器计数脉冲不变。例如:

当(F6, F5, F4, F3, F2, F1, F0) = (0, 1, 0, 1, 0, 0, 1), 且为 00、20、40 秒时刻时, 寄存器计数脉冲变为 $32\,768 + (41 - 1) \times 2 = 32848$;

当(F6, F5, F4, F3, F2, F1, F0) = (1, 1, 1, 1, 1, 1, 0), 且为 00、20、40 秒时刻时, 寄存器计数脉冲变为 $32\,768 - (1 + 1) \times 2 = 32764$;

当(F6, F5, F4, F3, F2, F1, F0) = (0, 0, 0, 0, 0, 0, 1), 且为 00、20、40 秒时刻时, 寄存器计数脉冲保持为 32 768 不变。

因为每 20 秒增加或减少计数脉冲的最小个数为 2, 所以时钟调整寄存器的最小调整精度是: $2/(32\,768 \times 20) = 3.052 \times 10^{-6}$ 。

注意: 时钟调整电路仅调整时钟走时, 并不对晶振本身频率进行调整, 所以 32.768 kHz 脉冲输出没有变化。

(3) 控制寄存器 1 中, 当(AALE, BALE) = 00 时, 禁止定时 A 和定时 B 中断; 当(AALE, BALE) = 11 时, 允许定时 A 和定时 B 中断。

设置(SL2, SL1)位, 能够确定闹钟 A、闹钟 B、周期性中断、32 kHz 时钟脉冲是从 INTRA 还是从 INTRB 引脚输出。当(SL2, SL1) = 00 时, 闹钟 A、闹钟 B 和周期性中断从 INTRA 输出, 32 kHz 时钟脉冲从 INTRB 输出; 当(SL2, SL1) = 01 时, 闹钟 A、周期性中断从 INTRA 输出, 32 kHz 时钟脉冲、闹钟 B 从 INTRB 输出; 当(SL2, SL1) = 10 时, 闹钟 A、闹钟 B 从 INTRA 输出, 32 kHz 时钟脉冲、周期性中断从 INTRB 输出; 当(SL2, SL1) = 11 时, 闹钟 A 从 INTRA 输出, 32 kHz 时钟脉冲、闹钟 B、周期性中断从 INTRB 输出。

设置(CT2, CT1, CT0)可选择周期性中断方式。当(CT2, CT1, CT0) = 000 时, INTRA 和 INTRB 为高电平; 当(CT2, CT1, CT0) = 001 时, INTRA 和 INTRB 为低电平; 当(CT2, CT1, CT0) = 010 时, 为 2 Hz 脉冲方式; 当(CT2, CT1, CT0) = 011 时, 为 1 Hz 脉冲方式; 当(CT2, CT1, CT0) = 100 时, 为电平方式(每秒产生 1 次); 当(CT2, CT1, CT0) = 101 时, 为电平方式(每分钟产生 1 次); 当(CT2, CT1, CT0) = 110 时, 为电平方式(每小时产生 1 次); 当(CT2, CT1, CT0) = 111 时, 为电平方式(每月产生 1 次)。

(4) 控制寄存器 2 中, ($\overline{12}/24$)位是小时制选择位。 $(\overline{12}/24) = 0$ 表示 12 小时制; $(\overline{12}/24) = 1$ 表示 24 小时制。

(ADJ)位是 ± 30 秒调整位。(ADJ) = 0 时, 正常工作; (ADJ) = 1 时, 秒调整操作。若读该

位 = 0, 则表示晶振工作正常; 若读该位 = 1, 则表示晶振停止工作(不正常)。

(CLEN)位是 32 kHz 脉冲输出使能位。(CLEN) = 0 时, 允许 32 kHz 输出; (CLEN) = 1 时, 禁止 32 kHz 输出。

(CTFG)位是周期性中断标志位。(CTFG) = 0 时, 无周期性中断; (CTFG) = 1 时, 有周期性中断。

(AAFG, BAFG)位是闹钟 A 和闹钟 B 标志位。(AAFG, BAFG) = 0 时, 无报时中断; (AAFG, BAFG) = 1 时, 有报时中断。

4. 串行接口

SD2098A 通过 SCL 和 SDA 2 线式串行接口方式接收各种命令并读/写数据。2 线式串行接口协议完全符合 I²C 总线方式。

1) 开始条件

当 SCL 处于高电平时, SDA 由高电平变成低电平时构成一个开始条件。对 SD2098A 的所有操作均必须由开始条件开始。

2) 停止条件

当 SCL 处于高电平时, SDA 由低电平变成高电平时构成一个停止条件。此时 SD2098A 的所有操作均停止, 系统进入待机状态。

3) 数据传输

当 SCL 为低电平, 且 SDA 线电平变化时, 数据由 CPU 传输给 SD2098A; 当 SCL 为高电平, 且 SDA 线电平不变时, CPU 读取 SD2098A 发送来的数据; 当 SCL 为高电平, 且 SDA 线电平变化时, SD2098A 收到一个开始或停止条件。

4) 确认

数据传输以 8 位序列进行。SD2098A 在第九个时钟周期时将 SDA 置为低电平, 即送出一个确认信号(Acknowledge 位, 简称“ACK”), 表明数据已经收到。

5. 读/写操作说明

1) 写操作

“写操作”的顺序是: 开始条件、写入“64H”、“寄存器地址(高 4 位)+0H(低 4 位)”、“被写入的数据”、停止条件。

2) 读操作

“读操作”有三种方法。第一种为标准读法, 即: 开始条件、写入“64H”、“寄存器地址(高 4 位)+0H(低 4 位)”、开始条件、写入“65H”、“数据 1”、“数据 2”、…、停止条件; 第二种为缩短顺序读法, 即: 开始条件、写入“64H”、“寄存器地址(高 4 位)+4H(低 4 位)”、“数据 1”、“数据 2”、…、停止条件; 第三种为只能从 FH 开始读, 即: 开始条件、写入“65H”、“0xF 的数据”、“0x0 的数据”、…、停止条件。

1.2.2 应用电路与编程

SD2098A 实时时钟与 STC89C52 单片机的接口电路如图 1-5 所示。这是一个标准的 I²C 总线接口。总线从地址写是 64H, 读是 65H。时序可参考后面的图 1-11~图 1-13。

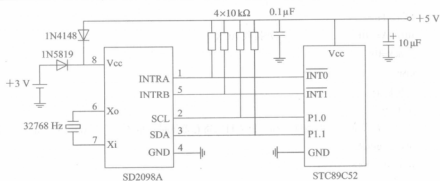


图 1-5 SD2098A 与单片机的连接图

用 C51 编写的相关“读”、“写”函数如下:

```
sbit SDA=P1^1          /*设置 I2C 数据读/写线*/
sbit SCL=P1^0          /*设置 I2C 时钟线*/
void Delay(void)        /*延时函数*/
```

```
{
    unsigned char i;
    for(i=0; i<2; i++); /*延时*/
}
```

/*功能: 向 SD2098A 的寄存器里写入一个字节(包括总线初始化、开始与终止条件)。

参数: reg 表示目的寄存器地址 0x00~0x0F;

dat 表示要写入的字节 */

```
void Write2098_i2c(unsigned char reg,unsigned char datai2c)
```

```
{
    unsigned char dat,b,i,ack; /*分别用于形参传递、8 位字长计数、延时等*/
    SDA=1; SCL=1;             /*初始化, 开始*/
    Delay(); SDA=0;
    Delay(); SCL=0;
    dat=0x64;                  /*发送写地址*/
    for(b=0; b<8; b++)
    {
        if(dat&0x80)           /*高位在前*/
            SDA=1;
        else
            SDA=0;             /*先准备好数据*/
        Delay(); SCL=1;        /*再在时钟线上发出一个正脉冲, 最后 SCL 被置 0*/
        Delay(); SCL=0;
        Delay(); dat<<=1;
    }
    SDA=1; Delay();           /*立即释放数据线*/
}
```

```

SCL=1;
if(SDA==0)          /*接收 ACK*/
    ack=0;
else
    ack=1;
SCL=0;
    /*发送寄存器地址。高 4 位为目的寄存器地址；低 4 位为 0000，为固定的写模式*/
dat=(reg<<4);
for(b=0; b<8; b++)
{
    if(dat&0x80)      /*高位在前*/
        SDA=1;
    else
        SDA=0;      /*先准备好数据*/
    Delay(); SCL=1;   /*延时，再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
    Delay(); SCL=0;
    Delay(); dat<<=1;
}
SDA=1; Delay();      /*立即释放数据线*/
SCL=1;
if(SDA==0)          /*接收 ACK*/
    ack=0;
else
    ack=1;
SCL=0;
dat=dat<<2;          /*发送数据*/
for(b=0; b<8; b++)
{
    if(dat&0x80)      /*高位在前*/
        SDA=1;
    else
        SDA=0;      /*先准备好数据*/
    Delay(); SCL=1;   /*延时，再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
    Delay(); SCL=0;
    Delay(); dat<<=1;
}
SDA=1; Delay();      /*立即释放数据线*/
SCL=1;
if(SDA==0)          /*接收 ACK*/
    ack=0;

```

```

        else
            ack=1;
        SCL=0; SDA=0;
        Delay(); SCL=1;
        Delay(); SDA=1;
    }

/*功能：从 SD2098A 的寄存器读出一个字节并作为返回值(包括开始与终止条件)。
参数：reg 表示目的寄存器地址 0x00~0x0F(SD2098A 内部的寄存器 s、m、h、...)*/
unsigned char Read2098_i2c(unsigned char reg)
{
    unsigned char datai2c;    /*读回的数据*/
    unsigned char dat, b, i, ack; /*分别用于形参传递、8 位字长计数、延时、取应答位*/
    SDA=1; SCL=1;            /*初始化*/
    Delay(); SDA=0;           /*延时、启动*/
    Delay(); SCL=0;           /*延时*/
    dat=0x64;                 /*发送写地址*/
    for(b=0; b<8; b++)
    {
        if(dat&0x80)          /*高位在前*/
            SDA=1;
        else
            SDA=0;            /*先准备好数据*/
        Delay(); SCL=1;        /*延时，再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
        Delay(); SCL=0;
        Delay(); dat<<=1;
    }
    SDA=1; Delay();           /*立即释放数据线*/
    SCL=1;
    if(SDA==0)                /*接收 ACK*/
        ack=0;
    else
        ack=1;
    SCL=0;

/*发送目的寄存器地址。高 4 位为目的寄存器地址；低 4 位为 0100，即缩短读模式 2*/
    dat=((reg<<4)|0x04);
    for(b=0; b<8; b++)
    {
        if(dat&0x80)          /*高位在前*/
            SDA=1;
        else

```



```

        SDA=0;          /*先准备好数据*/
        Delay(); SCL=1;  /*延时，再在时钟线上发出一个正脉冲，最后 SCL 被置 0*/
        Delay(); SCL=0;

    Delay(); dat<=1;
    }
    SDA=1; Delay();      /*立即释放数据线*/
    SCL=1;
    if(SDA==0)           /*接收 ACK*/
        ack=0;
    else
        ack=1;
    SCL=0;
    for(b=0; b<8; b++)    /*读数据*/
    {
        datai2c<=1;
        Delay(); SCL=1;  /*在 SCL 为高电平的情况下读 SDA*/
        if(SDA==0)
            datai2c&=0xfe; /*datai2c 末位清 0*/
        else
            datai2c|=0x01;  /*datai2c 末位置 1*/
        SCL=0;            /*时钟线置低*/
    }
    SCL=0; Delay();      /*发送 Nack，通知从机停止发送*/
    SDA=1; Delay();      /*准备好数据*/
    SCL=1; Delay();
    SCL=0; Delay();
    SDA=0; Delay();      /*停止条件*/
    SCL=1; Delay();
    SDA=1;
    return(datai2c);
}

void Write_SD2098A_inte(void) /*初始化函数*/
{
    Write2098_i2c(0x07, 0x00); /*不调整精度*/
    Write2098_i2c(0x08, 0x00); /*不报警*/
    Write2098_i2c(0x09, 0x00); /*不报警*/
    Write8025_i2c(0x0a, 0x00); /*不报警*/
    Write2098_i2c(0x0b, 0x00); /*不报警*/
    Write2098_i2c(0x0c, 0x00); /*不报警*/
    Write2098_i2c(0x0d, 0x00); /*不报警*/
}

```

```

        Write2098_i2c(0x0e, 0x00);
        Write2098_i2c(0x0f, 0x28);    /*24 小时制*/
    }

void Wdata_time2098(y,mo,d,h,m,s) /*写入时间与日历函数*/
unsigned char y,mo,d,h,m,s;
{
    Write2098_i2c(0x00, s);    /*秒*/
    Write2098_i2c(0x01, m);    /*分*/
    Write2098_i2c(0x02, h);    /*时*/
    Write2098_i2c(0x04, d);    /*日*/
    Write2098_i2c(0x05, mo);    /*月*/
    Write2098_i2c(0x06, y);    /*年*/
}

/*****读时钟*****/
void Read_date_time(y)    /*读年、月、日、时、分、秒函数*/
unsigned char y[];
{
    y[0]=Read2098_i2c(0x08);    /*年*/
    y[1]=Read2098_i2c(0x12);    /*月*/
    y[2]=Read2098_i2c(0x29);    /*日*/
    y[3]=Read2098_i2c(0x02);    /*时*/
    y[4]=Read2098_i2c(0x01);    /*分*/
    y[5]=Read2098_i2c(0x00);    /*秒*/
}

```

1.3 M41T0 低成本实时时钟

1.3.1 硬件与功能描述

M41T0 实时时钟 RAM 是一个低功耗串行计时器，它内置 32.768 kHz 频率校准电路。8 个寄存器用于实现时钟/日历的功能，配置了二进制编码的十进制格式输出形式。地址和数据的传输通过 2 线的双向总线连续传输，每次的“读”或“写”地址会自动增 1。M41T0 采用 8 引脚 SOIC 封装，适合各种小型电子设备使用。

1. 主要性能特点

- (1) 工作电压为 2.0~5.5 V；
- (2) 最低工作电流小于 130 μ A；
- (3) 工作温度范围为 -40~+85℃；
- (4) 年、月、日、星期、时、分、秒计数为 BCD 码输入/输出形式；
- (5) 具有闰年补偿换算功能；

- (6) 与 I^2C 总线接口兼容;
- (7) 具有特殊的软件可编程输出功能;
- (8) 具有振荡器停振检测功能。

2. 内部结构与引脚说明

M41T0 实时时钟的内部结构与引脚排列如图 1-6 所示。

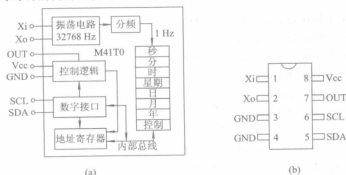


图 1-6 M41T0 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

图 1-6 中, X_i 、 X_o 为振荡器输入、输出端, 通常接 32 768 Hz 晶振; OUT 为开路输出脚; SDA 为串行数据(或地址)输入/输出端, 使用时要上拉 10 k Ω 电阻; SCL 为串行时钟输入脚, 一般上拉 10 k Ω 电阻。

3. 器件的操作协议

M41T0 器件在 SCL 和 SDA 串行总线上, 可按标准的 I^2C 总线协议访问内部(秒、分、时、星期、日、月、年和控制)8 个寄存器。该协议规定:

- (1) 当总线不忙(数据线和时钟线保持高电平)时, 数据传输可以启动。
- (2) 在数据传输期间, 当时钟线为高电平时, 数据线必须保持稳定(当时钟线为高电平时, 数据线上的变化将被认为是控制信号), 即①开始传输数据控制条件(当时钟线为高电平时, 数据线开始由高电平变为低电平); ②停止传输数据控制条件(当时钟线为高电平时, 数据线开始由低电平变为高电平); ③数据有效条件(在开始条件之后, 数据线状态有效, 数据线上的数据在时钟信号为高电平期间保持稳定, 在为低电平期间可以变化, 每一个数据位对应一个时钟脉冲)。
- (3) 每个数据的传输在开始条件下启动, 在停止条件下结束(时序可参考后面的图 1-11), 数据字节节的传输在开始条件和结束条件之间没有限制, 信息以字节宽度进行传输, 每个接收器的第 9 位为应答位。
- (4) 每一个 8 位字节都跟随一个应答位, 应答位以低电平的状态放入接收器的总线上, 而主控方需要产生一个与应答位相关的时钟脉冲。

4. 器件的读/写模式

1) 读模式

读模式的顺序是: 开始, 写入“D0H”、“寄存器地址”, 开始, 写入“DIH”, 读出“寄存器”数据 1、数据 2、…、数据 n, 停止。读数据时序如图 1-7 所示。

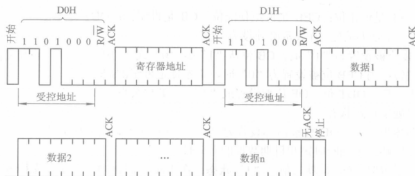


图 1-7 读数据时序

2) 写模式

写模式的顺序是：开始，写入“D0H”、“寄存器地址”，写入“寄存器”数据1、数据2、…、数据n，停止。写数据时序如图1-8所示。

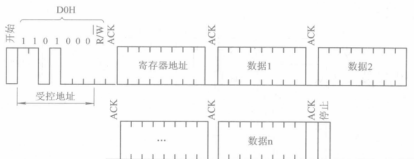


图 1-8 写数据时序

5. 寄存器的操作说明

M41T0 内部寄存器的地址与功能如表 1-4 所示。

表 1-4 M41T0 寄存器的地址与功能

寄存器地址	数 据 位								功能、范围与格式 输入/输出为 BCD 码	
	D7	D6	D5	D4	D3	D2	D1	D0		
0	ST	10 秒			秒				秒	00~59
1	OF	10 分			分				分	00~59
2	CEB	CB	10 时		时				世纪/时	0~1/00~23
3	x	x	x	x	x	星期			星期	01~07
4	x	x	10 天		天				天	01~31
5	x	x	x	10 月	月				月	01~12
6	10 年				年				年	00~99
7	OUT	0	x	x	x	x	x	x	控制	

表中: ST 是停止位; CEB 是世纪使能位; CB 是世纪位; OUT 是输出电平; OF 是振荡器失效位; x 是无关位; 0 表示必须设为“0”。

当 CEB 位被设置为“1”时, CB 位从“1”到“0”或从“0”到“1”的变化, 都将锁定世纪的翻转; 当 CEB 位被设置为“0”时, 将不锁定世纪的翻转。

初始上电后, OUT 位、OF 位被设置成“1”, ST 位被设置成“0”。其他所有寄存器在上电时处于任意状态。

当振荡器失效后, OF 位被内部设置为“1”, 这意味着振荡器停振或者计时的一些周期停止, 它能够用于判断时钟和日期数据的有效性。

第一次通电时, OF 位默认值为“1”状态; 在 V_{CC} 提供的电压不足以支持振荡器振荡时, ST 位会被置为“1”状态, 表示停机。

1.3.2 应用电路与编程

M41T0 与单片机的接口电路如图 1-9 所示。电路除 V_{CC} (系统)供电外, 还设计有“电池供电”电路, 以保证在系统掉电以后时钟能正常工作。时钟的“OUT”端连接有发光管, 以表示“系统电源正常”。

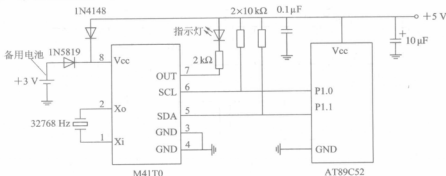


图 1-9 M41T0 的应用电路

对于图 1-9 所示的应用电路, 用 C51 编写有关函数如下:

```

sbit   SDA=P1^1          /*设置 I2C 数据读 / 写线*/
sbit   SCL=P1^0          /*设置 I2C 时钟线*/

void Delay(void)          /*延时函数*/
{
    unsigned char i;
    for (i=0; i<2; i++);
}

void wire_data(unsigned char m_data) /*写一个字节函数*/
{
    unsigned char b, dat, ack;
    dat=m_data;            /*发送 8 位数据或 8 位地址*/
    for(b=0; b<8; b++)
    {
        if(dat & 0x80)      /*高位在前*/

```

```

        SDA=1;
    else
        SDA=0;          /*先准备好数据*/
    Delay(); SCL=1;
    Delay(); SCL=0;
    Delay(); dat<=<=1;
}
SDA=1; Delay();        /*立即释放数据线*/
SCL=1;
if(SDA==0)              /*接收 ACK*/
    ack=0;
else
    ack=1;
SCL=0;
}

unsigned char read_data(unsigned char x)  /*读一个字节函数*/
{
    unsigned char b,datai2c;
    for(b=0; b<8; b++)
    {
        datai2c<=<=1;
        Delay(); SCL=1;      /*在 SCL 为高电平的情况下读 SDA*/
        if(SDA==0)
            datai2c &=0xfc;    /*datai2c 末位清 0*/
        else
            datai2c |=0x01;     /*datai2c 末位置 1*/
        SCL=0;                /*时钟线置低*/
    }
    if(x==0)                 /*x=0 时, 送 ACK 信号*/
    {
        SCL=0; Delay();      /*发送 Nack, 通知从机停止发送*/
        SDA=1; Delay();      /*准备好数据*/
        SCL=1; Delay();
        SCL=0; Delay();
    }
    return(datai2c);
}

void Write_M41T0_i2c(unsigned char m_data[]) /*写数据*/
{
    unsigned char i;

```

```

SDA=1; SCL=1;
Delay(); SDA=0;
Delay(); SCL=0;
wire_data(0xd0); /*发送受控地址 0xd0*/
wire_data(0x00); /*发送寄存器地址(秒地址)*/
for(i=0; i<8; i++)
    wire_data(m_data[i]); /*发送秒、分等 8 个数据*/
SDA=0; /*停止条件*/
Delay(); SCL=1;
Delay(); SDA=1;
}

void Read_M41T0_i2c(unsigned char m_data[]) /*读数据*/
{
    unsigned char i;
    SDA=1; SCL=1; /*开始条件*/
    Delay(); SDA=0;
    Delay(); SCL=0;
    wire_data(0xd0); /*发送受控地址 0xd0*/
    wire_data(0x00); /*发送寄存器地址(秒地址)*/
    SDA=1; SCL=1; /*重新开始一次*/
    Delay(); SDA=0;
    Delay(); SCL=0;
    wire_data(0xd1); /*发送读受控地址 0xd1*/
    for (i=0; i<6; i++) /*连续读 6 数据(有 ACK 信号)*/
        m_data[i]=read_data(0);
    m_data[6]=read_data(1); /*读第 6 个数据(无 ACK 信号)*/
    SDA=0; /*停止条件*/
    Delay(); SCL=1;
    Delay(); SDA=1;
}

void Write_M41T0(void) /*初始化*/
/*初始化设为: 2008 年 5 月 1 日、星期四、8 时 18 分 8 秒, 并点亮发光管*/
{ unsigned char Y[8]={0x8,0x18,0x8,0x4,0x1,0x5,0x8,0x80};
  Write_M41T0_i2c(Y);
}

void Reade_M41T0(void) /*读时钟(数据)*/
{ unsigned char Y[8]; /*将数据存放在 Y[8]中*/
  Read_M41T0_i2c(Y); /*数据在 Y 数组中*/
}

```

1.4 PCF8563 实时时钟

1.4.1 硬件与功能描述

PCF8563 是低功耗型 CMOS 实时时钟/日历芯片，具有可编程时钟输出、中断输出和掉电检测功能。所有的地址和数据通过串行 I²C 总线接口传送，最大总线速度为 400 kb/s。每次读/写数据后，内嵌的字地址寄存器会自动产生增量。

PCF8563 广泛应用于移动电话、便携仪器、传真机和多种电子产品中。

1. 主要性能特点

- (1) 工作电流的典型值为 0.25 μA ($V_{\text{CC}} = 3.0\text{ V}$ ，环境温度为 25℃ 时)；
- (2) 工作电压范围为 1.0~5.5 V；
- (3) 400 kHz 的 I²C 总线接口 ($V_{\text{CC}} = 1.8\sim 5.5\text{ V}$ 时)；
- (4) 可编程时钟输出频率为 32.768 kHz、1024 Hz、32 Hz 和 1 Hz；
- (5) I²C 总线从地址：读为 0A3H，写为 0A2H；
- (6) 具有世纪标志、报警、定时、掉电检测、片内电源复位和振荡器补偿电容等功能。
- (7) 中断为开漏输出。

2. 内部结构与引脚排列

PCF8563 的内部结构与引脚排列如图 1-10 所示。

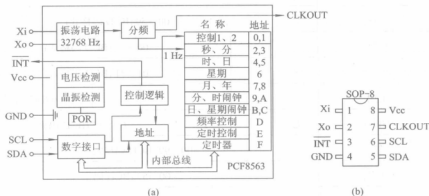


图 1-10 PCF8563 的内部结构与引脚排列

(a) 内部结构；(b) 引脚排列

PCF8563 采用 DIP-8、SOP-8 和 TSSOP-8 脚封装形式。其中：

- (1) Xi、Xo 为晶振的输入/输出端，通常接 32 768 Hz 晶振。
- (2) $\overline{\text{INT}}$ 为中断输出(漏极开路，低电平有效)。
- (3) SDA 是 I²C 接口数据输入/输出脚。
- (4) SCL 是 I²C 接口时钟输入脚。

(5) CLKOUT 是时钟输出端(漏极开路)。

PCF8563 集成有 16 个 8 位寄存器、1 个可自动增量的地址寄存器、1 个内置 32.768 kHz 的振荡器(带有内部集成的电容器)、1 个分频器、1 个可编程时钟输出、1 个定时器、1 个报警器、1 个掉电检测器和 1 个 400 kHz I²C 总线接口。

所有寄存器设计成可寻址的 8 位并行寄存器,但不是所有位都有用。地址 00H、01H 的两个寄存器为控制和状态寄存器,地址 02H~08H 的七个寄存器用于时钟计数器(秒~年计数器),地址 09H~0CH 的五个寄存器用于报警(或闹钟),地址 0DH 的寄存器用于控制 CLKOUT 管脚的输出频率,地址 0EH 和 0FH 的寄存器分别用于定时器控制和定时寄存器。秒、分钟、小时、日、月、年、分钟报警、小时报警、日报警寄存器的编码格式为 BCD 码,星期和星期报警寄存器的编码格式不是 BCD 码格式。

当一个(00H~0FH 中的一个)寄存器被读时,所有计数器的内容被锁存,因此,在传送条件下,可以禁止对时钟/日历芯片的错读。

3. 时间寄存器说明

PCF8563 器件的 16 个寄存器的含义如表 1-5 所示。

表 1-5 PCF8563 内部寄存器的地址与功能

[illegible]

1) “控制/状态寄存器1”中各位的含义

(1) TEST 位 = 0 是普通模式, TEST 位 = 1 是测试模式。

(2) STOP 位 = 0 是芯片时钟运行模式, STOP 位 = 1 表示计数时钟停止运行, 但 CLKOUT 输出 32 768 Hz 仍有效。

(3) TESTC 位 = 0 表示电源复位功能失效, TESTC 位 = 1 表示电源复位功能有效。

(4) 0 位表示为逻辑“0”。

2) “控制/状态寄存器2”中各位的含义

(1) 当 TI/TP 位 = 0 时, 若 TF 有效, 则 $\overline{\text{INT}}$ 有效(取决于 TIE 的状态); 当 TI/TP 位 = 1 时, $\overline{\text{INT}}$ 脉冲有效(取决于 TIE 的状态)。若 AF 和 AIE 都有效, 则 $\overline{\text{INT}}$ 一直有效。

(2) 当报警发生时, AF 被置逻辑 1; 在定时器倒数计数结束时, TF 被置逻辑 1。它们在被软件重写前一直保持原有值, 当定时器和报警中断都请求时, 中断源由 AF 和 TF 决定。若要清除一个标志位同时防止另一标志位被重写, 则可用逻辑指令 AND 屏蔽。

(3) 标志位 AIE 和 TIE 决定一个中断的请求有效或无效, 当 AF 或 TF 中一个为“1”时, 中断是 AIE 和 TIE 都置“1”时的逻辑或。

(4) AIE = 0 时, 报警中断无效; AIE = 1 时, 报警中断有效。TIE = 0 时, 定时器中断无效; TIE = 1 时, 定时器中断有效。

3) “秒、分钟和小时寄存器”中各位的含义

(1) “秒, VL”位 = 0 时, 保证准确的时钟/日历数据; “秒, VL 位” = 1 时, 不保证准确的时钟/日历数据。

(2) “—”位是无效数据。

(3) 其余位为 BCD 输入/输出格式。

4) “日、星期、月/世纪和年寄存器”中各位的含义

(1) “月/世纪, C”位 = 0 时指定世纪为 20xx; “月/世纪, C”位 = 1 时指定世纪为 19xx。

(2) “—”位是无效数据。

(3) 其余位为 BCD 输入/输出格式。

5) “报警寄存器”中各位的含义

当一个或多个报警寄存器写入合理的分钟、小时、日或星期数值并且它们相应的 AE (Alarm Enable) 位为逻辑 0, 以及这些数值与当前的分钟、小时、日或星期数值相等时, 标志位 AF (Alarm Flag) 被设置, AF 保存设置值, 直到被软件清除为止。AF 被清除后, 只有在时间增量与报警条件再次相匹配时才可再被设置。报警寄存器在它们的相应位 AE 置为逻辑 1 时将被忽略(相应报警无效)。

6) “CLKOUT 频率寄存器”中各位的含义

(1) FE 位 = 0 时, CLKOUT 输出被禁止并设成高阻抗; FE 位 = 1 时, CLKOUT 输出有效。

(2) 当 FD1、FD0 = 00 时, 为 32.768 kHz 输出; 当 FD1、FD0 = 01 时, 为 1024 Hz 输出; 当 FD1、FD0 = 10 时, 为 32 Hz 输出; 当 FD1、FD0 = 11 时, 为 1 Hz 输出。

7) “定时倒数计数寄存器”中各位的含义

定时器寄存器是一个 8 位的倒数计数定时器, 它由定时器控制器中的 TE 位决定有效或无效, 定时器的时钟也可以由定时器控制器选择。其他定时器功能如中断产生由控制/状态寄

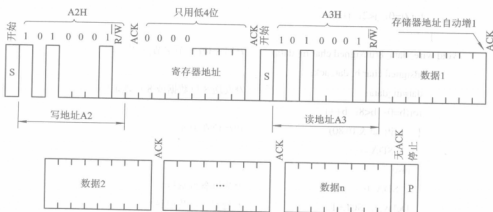


图 1-13 读时序

1.4.2 应用电路与编程

PCF8563 与单片机的接口电路如图 1-14 所示。电路除 +5 V 供电外，还备有“电池供电”电路，以保证系统掉电后时钟能正常工作。为了保证晶振频率准确，石英频率可以适当调整。

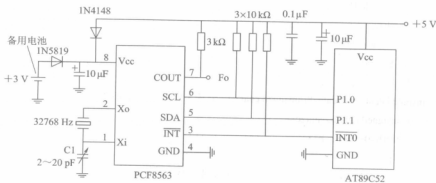


图 1-14 PCF8563 应用电路

方法 1：挑选最佳的电容 C1 (计算出所需的电容平均值)，用此电容通电后在第 7 脚上测出的频率应为 32.768 kHz，测出的频率值偏差取决于石英晶片、电容和器件之间的偏差 (平均为 $\pm 5 \times 10^{-6}$)。平均偏差可达 5 分钟/年。

方法 2：选择微调电容 C1，通过调整 Xi 管脚的微调电容使振荡器频率达到精确值，这时可测出通电后第 7 脚上的 32.768 kHz 信号。

对于图 1-14 所示的应用电路，用 C51 编写有关函数如下：

```
sbit SDA=P1^1 /*设置 I2C 数据读/写线*/
sbit SCL=P1^0 /*设置 I2C 时钟线*/

void Delay(void) /*延时函数*/
{ unsigned char j;
```

```

    for (j=0; j<2; j++){
    }
    void wire_data_8(unsigned char m_data) /*写 8 位(一个字节)函数*/
    { unsigned char b, dat, ack;
      dat=m_data; /*发送 8 位数据或 8 位地址*/
      for(b=0; b<8; b++){
        if(dat & 0x80) /*高位在前*/
          SDA=1;
        else
          SDA=0; /*先准备好数据*/
        Delay(); SCL=1; /*延时, 产生时钟*/
        Delay(); SCL=0;
        Delay(); dat<<=1; /*数据左移 1 位*/
      }
      SDA=1; Delay(); /*释放数据线, 延时, 准备接收 ACK*/
      SCL=1; /*产生时钟信号*/
      if(SDA==0) /*读 ACK 信号*/
        ack=0;
      else
        ack=1;
      SCL=0;
    }

    unsigned char read_data_8(unsigned char x) /*读 8 位(一个字节)函数*/
    { unsigned char b, datai2c;
      for(b=0; b<8; b++){
        {
          datai2c<<=1; /*数据先左移 1 位*/
          Delay(); SCL=1; /*延时, 使 SCL 变高电平*/
          if(SDA==0) /*在 SCL 为高电平的情况下读 SDA*/
            datai2c<&=0xfe; /*datai2c 末位清 0*/
          else
            datai2c|=0x01; /*datai2c 末位置 1*/
          SCL=0; /*数据读完, 时钟线置低*/
        }
      }
      if(x==0) /*x=0 时, 送 ACK 信号, 通知从机收到 1 位数据*/
      {
        SCL=0; Delay();
        SDA=1; Delay(); /*准备好数据, 延时*/
        SCL=1; Delay(); /*延时, 产生时钟*/
      }
    }

```

```

        SCL=0; Delay();
    }
    return(data12c);
}

void Write_P8563_i2c(unsigned char p_ab,p_data) /*写数据(一次写 1 个数据)*/
{
    SDA=1; SCL=1; Delay(); /*产生启动条件*/
    SDA=0; Delay(); SCL=0; /*SCL 后变低*/
    wire_data_8(0xA2); /*发送从地址 0xA2*/
    wire_data_8(p_ab); /*发送寄存器地址 p_ab*/
    wire_data_8(m_data); /*发送数据*/
    SDA=0; Delay(); /*产生停止条件*/
    SCL=1; Delay();
    SDA=1; /*SDA 后变高*/
}

void Read_P8563_i2c(unsigned char m_data[]) /*读数据(一次读 16 个)*/
{
    unsigned char i;
    SDA=1; SCL=1; /*产生开始条件*/
    Delay(); SDA=0;
    Delay(); SCL=0; /*SCL 后变低*/
    wire_data_8(0xA2); /*发送从地址 0xA2*/
    wire_data_8(0x00); /*发送寄存器地址(00H)*/
    SDA=1; SCL=1; /*重新产生开始条件*/
    Delay(); SDA=0;
    Delay(); SCL=0;
    wire_data_8(0xA3); /*发送读受控地址 0xA3*/
    for (i=0; i<15; i++) /*连续读 15 个数据(有 ACK 信号)*/
        m_data[i]=read_data_8(0);
    m_data[15]=read_data_8(1); /*读第 16 个数据(无 ACK 信号)*/
    SDA=0; /*产生停止条件*/
    Delay(); SCL=1;
    Delay(); SDA=1;
}

void Write_PCF8563(void) /*初始化*/
/*初始化设为: 2008 年 5 月 3 日、星期六、17 时 01 分 02 秒*/
{
    Write_P8563_i2c(0x00,p_0x00); /*正常工作模式*/
    Write_P8563_i2c(0x01,p_0x00); /*不中断、不报警*/
    Write_P8563_i2c(0x02,p_0x02); /*秒设为 02 秒*/
}

```

```

Write_P8563_i2c(0x03,p_0x01);    /*分设为 01 分钟*/
Write_P8563_i2c(0x04,p_0x17);    /*时设为 17 时*/
Write_P8563_i2c(0x05,p_0x03);    /*日设为 3 号*/
Write_P8563_i2c(0x06,p_0x06);    /*星期设为 6*/
Write_P8563_i2c(0x07,p_0x05);    /*月设为 5 月*/
Write_P8563_i2c(0x08,p_0x08);    /*年设为 08 年*/
Write_P8563_i2c(0x09,p_0x80);    /*分钟报警无效*/
Write_P8563_i2c(0x0A,p_0x80);    /*小时报警无效*/
Write_P8563_i2c(0x0B,p_0x80);    /*日报警无效*/
Write_P8563_i2c(0x0C,p_0x80);    /*星期报警无效*/
Write_P8563_i2c(0x0D,p_0x80);    /*CLKOUT 输出 32768 Hz 频率*/
Write_P8563_i2c(0x0E,p_0x00);    /*倒计时无效*/
Write_P8563_i2c(0x0F,p_0x01);    /*倒计时为 n=1*/
}

void Reade_PCF8563(void)          /*读时钟(数据)*/
{ unsigned char Y[16];           /*将数据存放在 Y[16]中*/
  Read_P8563_i2c(Y);             /*数据在 Y 数组中*/
}

```

1.5 DS1302 低功耗自带 RAM 实时时钟

1.5.1 硬件与功能描述

DS1302 是一种高性能、低功耗的串行接口专用实时时钟芯片, 附加 31 字节静态 RAM, 采用串行接口与 CPU 进行同步通信, 并可采用突发方式一次传送多个字节时钟信号和 RAM 数据。实时时钟可提供秒、分、时、日、星期、月和年, 一个月小于 31 天时可以自动调整, 并具有闰年补偿功能。

该器件可设计成能在非常低的功耗下工作, 消耗小于 1 微瓦的功率便能保存数据和时钟信息。DS1302 是 DS1202 的升级产品, 除了 DS1202 基本的慢速充电功能外, DS1302 还有用于主电源和备份电源的双电源结构。

DS1302 用于数据记录, 特别是对某些具有特殊意义的数据点, 能实现数据与出现该数据的时间同步记录的功能。

1. 主要性能特点

- (1) 低工作电流: 典型值为 $0.3 \mu\text{A}$ ($V_{\text{CC}} = 2.0 \text{ V}$, 环境温度为 25°C);
- (2) 电源电压范围: $2.0 \sim 5.5 \text{ V}$, 有备用电源接口;
- (3) 内部有可编程的充电电路;
- (4) 实时时钟: 秒、分、时、日、星期、月和年;
- (5) 内部存储器: 31 字节 ($31 \times 8 \text{ bit}$);

- (6) 7个附加字节的高速缓存存储器;
- (7) 3线串行通信, 兼容 TTL 和 CMOS 电平;
- (8) 工作温度: $0\sim+70^{\circ}\text{C}$ (民品级), $-45\sim+85^{\circ}\text{C}$ (工业级)。

2. 内部结构与引脚说明

DS1302 的内部结构与引脚排列如图 1-15 所示。它主要由输入移位寄存器、指令控制逻辑、振荡电路、实时时钟核心单元、内部 RAM、充电电路等组成。

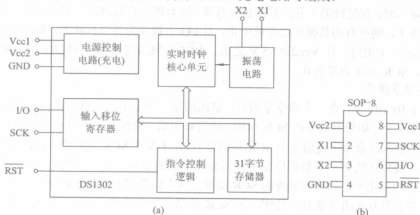


图 1-15 DS1302 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

DS1302 的外形封装采用 DIP-8 或 SOP-8。其中:

- (1) X1、X2 是外接 32 768 Hz 晶振的输入/输出端, 通常外接补偿电容。
- (2) Vcc2 是主电源, 一般接 $+3\sim+5\text{ V}$ 电源。
- (3) Vcc1 是备用电源, 一般接 $+3.6\text{ V}$ 可充电电池或 $+3\text{ V}$ 干电池。
- (4) I/O 是串行数据输入/输出端, 在 SCK 同步时钟驱动下, 传输内部数据。
- (5) SCK 是串行时钟输入端。
- (6) $\overline{\text{RST}}$ 是复位输入端, 低电平有效。
- (7) GND 是参考地。

3. 使用说明

DS1302 采用 3 线(I/O、SCK、 $\overline{\text{RST}}$)与控制器(单片机)通信。DS1302 提供秒、分、时、日、星期、月和年等信息。对于小于 31 天的月, 月末的日期自动进行调整, 还包括了闰年校正功能。时钟的运行可以采用 24 小时或带 AM/PM 的 12 小时格式。数据可以以每次一个字节或多达 31 字节进行数据传输。

1) 命令字节

每一数据传送由命令字节开始。最高有效位 MSB(位 7)必须为逻辑 1。如果它为 0, 则禁止与 DS1302 交换数据。位 6(R/C)为逻辑 0 时选择的是时钟/日历数据, 为逻辑 1 时选择的是 RAM 数据。位 5~位 1(A4~A0 地址)是 5 位地址信号(日历寄存器或 RAM 地址)。位 0 是读/写(R/ $\overline{\text{W}}$)控制位, 当 $\text{R}/\overline{\text{W}} = 0$ 时是写操作, 当 $\text{R}/\overline{\text{W}} = 1$ 时是读操作。命令字节总是从最低有效 LSB 位(位 0)开始输入。其格式如表 1-6 所示。

表 1-6 DS1302 的命令格式

D7	D6	D5	D4	D3	D2	D1	D0
1	R/C	A4	A3	A2	A1	A0	R/W

2) 复位和时钟控制

$\overline{\text{RST}}$ 输入有两种功能。首先, $\overline{\text{RST}}$ 接通控制逻辑, 允许地址/命令序列送入移位寄存器; 其次, $\overline{\text{RST}}$ 提供了中止单字节或多字节数据传送的功能(类似片选信号)。

数据输入时, 在时钟的上升沿数据必须有效, 而数据位在时钟的下降沿输出。如果 $\overline{\text{RST}}$ 输入为低电平, 则所有的数据传送将被中止, 且 I/O 引脚变为高阻抗状态。数据传送时序如图 1-16 所示。上电时, 在 $V_{cc2} \geq 2.5 \text{ V}$ 之前, $\overline{\text{RST}}$ 必须为逻辑 0。当把 $\overline{\text{RST}}$ 驱动至逻辑 1 的状态时, SCK 必须为逻辑 0。

3) 单字节操作

如图 1-16(a)所示, 在一个命令字节后, 紧跟的是一个字节的数据。在 SCK 周期的上升沿输入数据字节。如果有额外的 SCK 周期, 则将被忽略。数据从最低位 0 开始输入或输出。

跟随在输入读命令字节的 8 个 SCK 周期之后, 在下 8 个 SCK 周期的下降沿输出数据字节。注意, 被传送的第一个数据位发生在写命令字节的最后一位之后的第一个下降沿。只要 $\overline{\text{RST}}$ 保持为高电平, 如果有额外的 SCK 周期, 则它们将重新发送数据字节。这一操作具有连续的多字节方式的读能力。另外, 在 SCK 的第一个上升沿, I/O 引脚为三态。数据从位 0 开始输出。

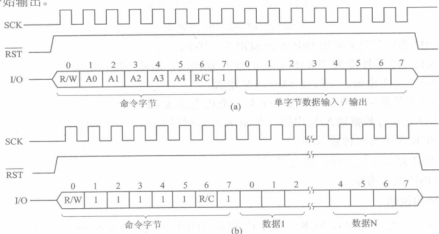


图 1-16 操作时序

(a) 单字节时序; (b) 多字节时序

4) 多字节方式

通过对地址 31(十进制)寻址, 可以把时钟/日历或 RAM 寄存器设定为多字节方式。如表 1-6, 位 6 规定时钟或 RAM, 而位 0 规定读或写。在时钟/日历寄存器中的地址 9~31 或 RAM 寄存器中的地址 31 不能存储数据。在多字节方式中读或写从地址 0 的位 0 开始。

当以多字节方式写时钟寄存器时, 必须按数据传送的次序写最前面的 8 个寄存器。但是, 当以多字节方式写 RAM 时, 为了传送数据, 不必写所有的 31 个字节。不管是否写完了全部 31 个字节, 所写的每一个字节都将送至 RAM。时序如图 1-16(b)所示。

5) DS1302 内部寄存器

DS1302 内部“秒~年”寄存器地址(命令)及数据寄存器的分配情况如表 1-7 所示。

表 1-7 内部资源分配

名 称	时钟命令字节(R/C=0)								数 据 字 节								
秒	1	0	0	0	0	0	0	R/W	PH	10 秒				秒			
分	1	0	0	0	0	0	1	R/W	0	10 分				分			
时	1	0	0	0	0	1	0	R/W	12/24	10/A/P		时		时			
日	1	0	0	0	0	1	1	R/W	0	0	10 日		日				
月	1	0	0	0	1	0	0	R/W	0	0	10 月		月				
星期	1	0	0	0	1	0	1	R/W	0	0	0	0	0	星期			
年	1	0	0	0	1	1	0	R/W	10 年				年				
保护	1	0	0	0	1	1	1	R/W	WP	0	0	0	0	0	0	0	0
充电	1	0	0	1	0	0	0	R/W	TC	TC	TC	TC	DS	DS	RS	RS	RS
多字节	1	0	1	1	1	1	1	R/W	—	—	—	—	—	—	—	—	—
存储器(RAM)地址命令(R/C=1)									数据字节								
RAM0	1	1	0	0	0	0	0	R/W	x	x	x	x	x	x	x	x	x
RAM0	1	1	0	0	0	0	1	R/W	x	x	x	x	x	x	x	x	x
...									...								
RAM30	1	1	1	1	1	1	0	R/W	x	x	x	x	x	x	x	x	x
多字节	1	1	1	1	1	1	1	R/W	—	—	—	—	—	—	—	—	—

对表 1-7 的说明如下:

(1) 时钟/日历与时钟暂停。时钟/日历(秒~年)包含在 7 个寄存器中。每个寄存器的数据用二十进制(BCD)码表示。秒寄存器的位 7 定义为时钟暂停位(PH)。当此位(PH)设置为逻辑 1 时, 时钟振荡器停止, DS1302 进入低功耗的备用方式, 其电源消耗小于 100 nA; 当把此位置逻辑 0 时, 时钟将重新启动。

(2) 12/24/AM/PM/方式。小时寄存器的位 7 定义为 12/24 小时方式选择位。当它为高电平时, 选择 12 小时方式。在 12 小时方式下, 位 5 是 AM/PM 位, 此位为逻辑高电平表示 PM。在 24 小时方式下, 位 5 是第 2 个 10 小时位(20~23 时)。

(3) 写保护寄存器。写保护寄存器的位 7(WP)是写保护位。位 6~位 0 置为 0, 在读操作时总是读出 0。在对时钟或 RAM 进行写操作之前, 位 7(WP)必须为 0。当它为高电平时, 将禁止对任何其他寄存器进行写操作(写保护)。

(4) 慢速充电寄存器。这个寄存器控制 DS1302 的慢速充电特性。图 1-17 是简化的慢速充电原理电路。慢速充电选择位(7~4, TC)控制慢速充电器的电流。为了防止偶然使之工作, 于是只有在 1010 模式才能使慢速充电器工作, 其他所有模式将禁止慢速充电器工作。DS1302 上电时, 慢速充电器被禁止。二极管选择位 DS(位 2、位 3)用来选择是一个二极管还是两个二极管连接在 Vcc1 与 Vcc2 之间。如果 DS 为 01, 那么选择一个二极管; 如果 DS 为 10, 则选择两个二极管。如果 DS 为 00 或 11, 那么充电器被禁止, 与 TC 无关。RS 位(位 0、位 1)选择连接在 Vcc1 与 Vcc2 之间的限流电阻。

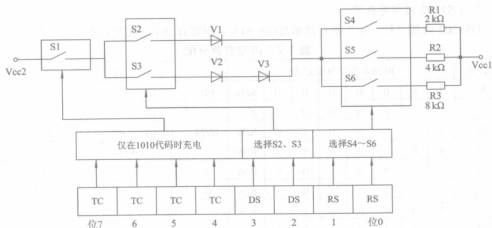


图 1-17 DS1302 可编程慢速充电电路

(5) 时钟/日历多字节方式。时钟/日历命令字节可规定多字节方式。在此方式下，最先 8 个时钟/日历寄存器可以从地址 0 的第 0 位开始连续读或写。

当指定写时钟/日历的多字节方式时，如果写保护位(WP)设置为高电平，那么没有数据会传送到 8 个时钟/日历寄存器(包括控制寄存器)中的任何一个。在多字节方式下，慢速充电器是不可访问的。

(6) 单字节与多字节 RAM 方式。静态 RAM 是 RAM 地址空间可按地址读/写的存储器(31×8 字节)。也可设置成多字节工作方式，在此方式下，可以从地址 0 的第 0 位开始顺序读或写 31 个字节的 RAM 寄存器。

(7) 晶振的选择。32 768 Hz 的晶振可通过引脚 2 和 3(X1 和 X2)直接连接至 DS1302，所选用晶振规定的负载电容量一般为 6 pF。

然而，在选用晶振时如果仅仅注意了晶振的额定频率值，而忽视了晶振的负载电容大小，或者所选 32 768 Hz 晶振与负载电容不一致，则会影响晶振的起振或导致振荡频率的偏移。一般可通过下述方法解决：

① 当所选的晶振负载电容不是 6 pF 时，可以采用增加辅助电容的方法提高或降低 DS1302 振荡器的电容负载，使之与晶体所需的电容值匹配。

已知晶体的负载电容为 C_1 ，若 $C_1 < 6 \text{ pF}$ ，则可以增加一个并联电容 C_s 以产生所需的总负载电容 C_1 ，即 $C_1 = 6 \text{ pF} + C_s$ ；若 $C_1 > 6 \text{ pF}$ ，则可以在晶体的一端增加一个串联电容 C_s ，以产生所需的负载电容 C_1 ，即 $1/C_1 = 1/6 \text{ pF} + 1/C_s$ ，通过计算即可得出应增加的辅助电容大小。

② 若在使用前对晶体的负载电容并不知道，则可通过测定晶体振荡频率的方法确定该晶体的负载电容。

对于晶体振荡器来说，其振荡频率与负载电容之间的关系是确定的。以 DS1302 使用的 32 768 Hz 晶振为例，当它工作于所要求的负载电容时，能较准确地产生 32 768 Hz 的频率；当它的负载电容小于 6 pF 时，其振荡频率会正向偏移；当它的负载电容大于 6 pF 时，其振荡频率就会负向偏移。因此，对于未知负载电容的晶体，应首先采用实验的方法，在其两

端加入辅助电容使晶体起振,然后用频率计测出振荡频率。若测得频率大于 32 768 Hz,则说明负载电容偏小;若测得频率小于 32 768 Hz,则说明负载电容偏大。逐步调整辅助电容,最终使振荡频率尽可能接近 32 768 Hz,则此时晶体端所接负载电容的总和就是适合该晶体的负载电容。

(8) 电源控制。Vcc1 是为 DS1302 提供的备用电源, 一般可接 3 V 干电池或 3.6 V 蓄电池(以便充电)。

Vcc2 是为 DS1302 提供的主电源, 在这种运行方式中 Vcc2 给芯片供电, 并通过内部为备用电源充电, 以便在没有主电源的情况下能保证时间信息及数据不被破坏。

DS1302 由 V_{cc1} 或 V_{cc2} 两者中较大者供电。当 V_{cc2} 大于 $V_{cc1} + 0.2\text{ V}$ 时, V_{cc2} 给 DS1302 供电; 当 V_{cc2} 小于 V_{cc1} 时, DS1302 由 V_{cc1} 供电。

1.5.2 应用电路与编程

1. 应用电路

图 1-18 是 DS1302 在 $\Gamma^2\text{C}$ 、SPI 存储器实验系统中的应用接法。其中, 单片机的 P2.7、P1.2 和 P1.3 分别接 DS1302 的 I/O、 $\overline{\text{RST}}$ 和 SCK。图中备用电源接的是 3 V 的干电池, 因此二极管 V2 的作用是保护电池(或在编程时不用给电池充电)。

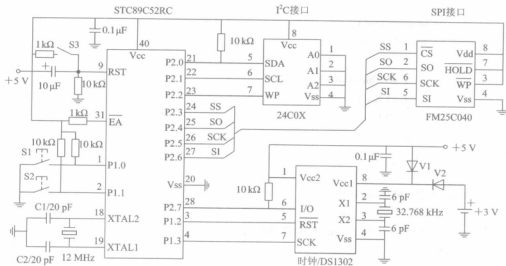


图 1-18 DS1302 的应用电路

2. 软件编程

通过对 DS1302 原理和使用方法的掌握, 结合 C51 语言的编程特点, 所实现的程序源代码如下:

```
#include <reg51.h>
sbit T_IO  = P2^7;           /*实时时钟数据线引脚*/
sbit T_RST = P1^2;           /*实时时钟复位线引脚*/
```

```

sbit T_CLK = P1^3; /*实时时钟的时钟线引脚*/
/*函数声明*/
void RTInputByte(unsigned char d); /*输入 1 B*/
unsigned char RTOutputByte(void); /*输出 1 B*/
void W1302(unsigned char ucAddr, unsigned char ucDa); /*向 DS1302 写入数据*/
unsigned char R1302(unsigned char ucAddr); /*读取 DS1302 某地址的数据*/
void Set1302(unsigned char *pClock); /*设置时间*/
void Get1302(unsigned char curtime[]); /*读取 1302 当前时间*/
/*****

```

函数名: RTInputByte()

功能: 实时时钟写入一字节

说明: 往 DS1302 写入 1 byte 数据(内部函数)

入口参数: d 为写入的数据

返回值: 无

*****/

```

void RTInputByte(unsigned char d)
{

```

```

    unsigned char i;

```

```

    for(i=8; i>0; i--)
    {

```

```

        T_IO=(bit)(d&0x01);

```

```

        T_CLK = 1;

```

```

        T_CLK = 0;

```

```

        d = d >> 1;
    }
}

```

/*函数名: RTOutputByte()

功能: 实时时钟读取一字节

说明: 从 DS1302 读取 1 byte 数据(内部函数)

入口参数: 无

返回值: temp */

```

unsigned char RTOutputByte(void)
{

```

```

    unsigned char i;

```

```

    unsigned char temp=0;

```

```

    for(i=8; i>0; i--)
    {

```

```

        temp=temp>>1;

```

```

        if(T_IO==1) temp=temp|0x80;
    }
}

```

```

        T_CLK = 1;
        T_CLK = 0;
    }
    return(temp);
}

```

/*功 能：往 DS1302 写入数据

说 明：先写地址，后写命令/数据(内部函数)

调 用：RTInputByte()

入口参数：ucAddr 为 DS1302 地址，ucData 为要写的

返 回 值：无 */

void W1302(unsigned char ucAddr, unsigned char ucData)

```

{
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    RTInputByte(ucAddr); /*地址，命令*/
    RTInputByte(ucData); /*写 1 byte 数据*/
    T_CLK = 1;
    T_RST = 0;
}

```

/*函 数 名：R1302()

功 能：读取 DS1302 某地址的数据

说 明：先写地址，后读命令/数据(内部函数)

调 用：RTInputByte(), RTOutputByte()

入口参数：ucAddr 为 DS1302 地址

返 回 值：ucData 为读取的数据 */

unsigned char R1302(unsigned char ucAddr)

```

{
    unsigned char ucData;
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    RTInputByte(ucAddr); /*地址，命令*/
    ucData = RTOutputByte(); /*读 1 byte 数据*/
    T_CLK = 1;
    T_RST = 0;
    return(ucData);
}

```

/*函 数 名：W1302T_String()

功能: 往 DS1302 写入时钟数据(多字节方式)

说明: 先写地址, 后写命令/数据

调用: RTInputByte()

入口参数: pWClock 为时钟数据地址, 格式为: 秒 分 时 日 月 星期 年 控制

8 byte(BCD 码)1B 1B 1B 1B 1B 1B 1B 1B

返回值: 无 */

void W1302T_String(unsigned char *pWClock)

```
{
    unsigned char i;
    W1302(0x8e, 0x00);          /*控制命令, WP=0, 写操作*/
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    RTInputByte(0xbe);          /*0xbe: 时钟多字节写命令*/
    for (i = 8; i > 0; i--)      /*8B = 7B 时钟数据 + 1B 控制*/
    {
        RTInputByte(*pWClock);  /*写 1 byte 数据*/
        pWClock++;
    }
    T_CLK = 1;
    T_RST = 0;
}
```

/*函数名: R1302T_String()

功能: 读取 DS1302 时钟数据

说明: 先写地址/命令, 后读数据(时钟多字节方式)

调用: RTInputByte(), RTOutputByte()

入口参数: pRClock 为读取时钟数据地址, 格式为: 秒 分 时 日 月 星期 年

7 byte(BCD 码)1B 1B 1B 1B 1B 1B 1B

返回值: 无 */

void R1302T_String(unsigned char *pRClock)

```
{
    unsigned char i;
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    RTInputByte(0xbf);          /*0xbf: 时钟多字节读命令*/
    for (i = 8; i > 0; i--)
    {
        *pRClock = RTOutputByte(); /*读 1 byte 数据*/
    }
}
```

```

        pRClock++;
    }
    T_CLK = 1;
    T_RST = 0;
}

```

/*函数名: W1302R_String()

功能: 往 DS1302 寄存器写入数据(多字节方式)

说明: 先写地址, 后写数据(寄存器多字节方式)

调用: RTInputByte()

入口参数: pWReg 为寄存器数据地址

返回值: 无 */

```

void W1302R_String(unsigned char *pWReg)
{
    unsigned char i;
    W1302(0x8e, 0x00); /*控制命令, WP=0, 写操作*/
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;
    RTInputByte(0xfe); /*0xfe: 时钟多字节写命令*/
    for (i=31; i>0; i--) /*31 byte 寄存器数据*/
    {
        RTInputByte(*pWReg); /*写 1B 数据*/
        pWReg++;
    }
    T_CLK = 1;
    T_RST = 0;
}

```

/*函数名: R1302R_String()

功能: 读取 DS1302 寄存器数据

说明: 先写地址, 后读命令/数据(寄存器多字节方式)

调用: RTInputByte(), RTOutputByte()

入口参数: pRReg 为寄存器数据地址

返回值: 无 */

```

void R1302R_String(unsigned char *pRReg)
{
    unsigned char i;
    T_RST = 0;
    T_CLK = 0;
    T_RST = 1;

```



```

RTInputByte(0xff);          /*0xff: 时钟多字节读命令*/
for (i=31; i>0; i--)        /*31 byte 寄存器数据*/
{
    *pRReg = RTOutputByte(); /*读 1 byte 数据*/
    pRReg++;
}
T_CLK = 1;
T_RST = 0;
}

/*函数名: Set1302()
功 能: 设置初始时间
说 明: 先写地址, 后读命令/数据(寄存器多字节方式)
调 用: W1302()
入口参数: pClock 为设置时钟数据地址, 格式为: 秒 分 时 日 月 星期 年
7 byte (BCD 码) 1B 1B 1B 1B 1B 1B 1B
返 回 值: 无 */
void Set1302(unsigned char *pClock)
{
    unsigned char i;
    unsigned char ucAddr = 0x80;
    W1302(0x8e, 0x00);      /*控制命令, WP=0, 写操作*/
    for(i=7; i>0; i--)
    {
        W1302(ucAddr, *pClock); /*秒 分 时 日 月 星期 年*/
        pClock++;
        ucAddr +=2;
    }
    W1302(0x8e, 0x80);      /*控制命令, WP=1, 写保护*/
}

/*函数名: Get1302()
功 能: 读取 DS1302 当前时间
说 明:
调 用: R1302()
入口参数: ucCurtime 为保存当前时间地址。当前时间格式为: 秒 分 时 日 月 星期 年
7 byte (BCD 码) 1B 1B 1B 1B 1B 1B 1B
返 回 值: 无 */
void Get1302(unsigned char ucCurtime[])
{
    unsigned char i;

```

```
unsigned char ucAddr = 0x81;
for (i=0; i<7; i++)
{
    ucCurtime[i] = R1302(ucAddr); /*格式为: 秒 分 时 日 月 星期 年*/
    ucAddr += 2;
}
)
/*****测试程序*****/
void main(void) /*主函数部分*/
{
    unsigned char Current_time[7];
    Get1302(Current_time); /*读取时间, 保存到 Current_time 数组中*/
    while(1); /*停止*/
}
```

第2章 集成传感器的使用与编程

2.1 DS18B20 数字温度传感器

2.1.1 硬件与功能描述

DS18B20 数字温度传感器是一种改进型智能温度传感器,与传统的热电阻、热电偶等测温元件相比,它能直接读出被测温度值,并且可根据实际要求通过简单的编程实现 9~12 位数字值读出。DS18B20 采用单线与主机传送信息,因此从主机 CPU 到 DS18B20 仅需一条线(除地线外)。DS18B20 的电源线可以由数据线本身提供,而不需要外部电源(采用寄生电源模式)。每一个 DS18B20 在出厂时已经给定了唯一的序号,因此任意多个 DS18B20 可以连接在同一条单线总线(又称单总线)上,实现不同地方的多点温度测量。DS18B20 的测量范围为 $-55\sim+125^{\circ}\text{C}$,且在 $-10\sim+85^{\circ}\text{C}$ 内精度为 $\pm 0.5^{\circ}\text{C}$ 。

每个 DS18B20 包括一个唯一的 64 位长的序列号,该序号值存放在 DS18B20 内部的 ROM(只读存储器)中。64 位序列中,最低 8 位是产品类型的编码(DS18B20 编码均为 10H),接着的 48 位是每个器件唯一的序号(48 位),最高 8 位是前 56 位的 CRC(循环冗余校验)码。DS18B20 中还有用于存储测量值的两个 8 位存储器, RAM 编号为 0 号和 1 号。1 号存储器存放温度值的符号,如果温度为负,则 1 号 8 位存储器全为 1,否则全为 0。0 号存储器用于存放温度值的补码, LSB(最低位)的 1 表示 0.5°C 。将存储器中的二进制数求补后再转换成十进制数,并除以 2 就得到被测温度值($-55\sim+125^{\circ}\text{C}$)。

1. 主要性能特点

- (1) 独特的单线接口仅需一个引脚进行通信;
- (2) 每个器件有唯一的 64 位序列号,可实现多点分布式温度的测量;
- (3) 无需外部器件,零待机功耗;
- (4) 可通过数据线供电,电压范围为 $+3.0\sim+5.5\text{ V}$;
- (5) 温度以 9~12 位数字值读出,测量范围为 $-55\sim+125^{\circ}\text{C}$;
- (6) 通常在 0.75 s 内可将温度转换成数字量;
- (7) 可定义非易失性温度报警设置;
- (8) 具有电源接反输入保护功能。

2. 内部结构与引脚说明

DS18B20 的内部结构与引脚排列如图 2-1 所示。

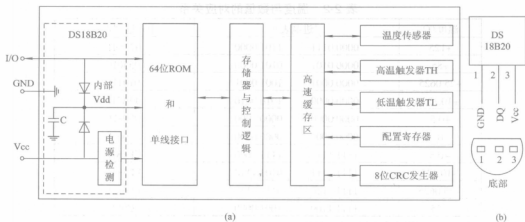


图 2-1 DS18B20 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

DS18B20 的引脚采用 TO-92 三脚封装。其中:

- (1) GND 是接地脚。
- (2) DQ 是数据输入/输出引脚(漏极开路)。当工作于 3 线方式(GND、Vcc 和 DQ)时, DQ 是输入/输出信号线;当工作于单线操作(GND 和 DQ)时, DQ 既是数据线又是电源线(通过寄生电源模式提供电源)。
- (3) Vcc 是电源脚。当用于 3 线方式时, Vcc 是电源脚;当用于单线方式时, Vcc 脚必须接地。

3. 温度测量

DS18B20 的核心功能是直接读出温度值的数字量。数字温度传感器的测量位数可编程为 9 位、10 位、11 位和 12 位, 分别以 0.5°C 、 0.25°C 、 0.125°C 和 0.0625°C 增量递增。在上电状态下默认的精度为 12 位。DS18B20 启动后保持低功耗等待状态, 当需要执行温度测量和 A/D 转换时, 总线必须发出 44H 命令。此后, 产生的温度数据以两个字节形式被存储到高速存储器中, 并继续保持等待状态。

当 DS18B20 由外部电源供电时, 总线控制器在温度转换指令之后发起“读时序”, DS18B20 在温度转换中返回 0, 转换结束后返回 1。如果 DS18B20 由寄生电源供电, 则除非在进入温度转换时, 总线被一个强上拉拉高, 否则将不会有返回位。

温度寄存器的格式如表 2-1 所示。其中, D15~D8 为高字节, D7~D0 为低字节。最高位为符号位(S), 负温度 S=1, 正温度 S=0。

表 2-1 温度寄存器的格式

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
S	S	S	S	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

温度与数值的对应关系如表 2-2 所示。

表 2-2 温度与数值的对应关系

温度/℃	二进制表示		十六进制表示
+125	0000 0111	1101 0000	07D0H
+85	0000 0101	0101 0000	0550H
+25.0625	0000 0001	1001 0001	0191H
+10.125	0000 0000	1010 0010	00A2H
+0.5	0000 0000	0000 1000	0008H
0	0000 0000	0000 0000	0000H
-0.5	1111 1111	1111 1000	FFF8H
-10.125	1111 1111	0101 1110	FF5EH
-25.0625	1111 1110	0110 1111	FE6FH
-55	1111 1100	1001 0000	FC90H

注：上电时温度寄存器的默认值为+85℃。

4. 报警操作信号

DS18B20 每完成一次温度测量就与用户预设存储在寄存器 TH 和 TL 中的数值作比较。TH 和 TL 为 8 位寄存器时，位 7 是符号位(S=0，表示正数；S=1，表示负数)，位 6~位 0 表示温度值($2^6 \sim 2^0$)。在比较后，如果测量值高于 TH 或低于 TL(报警条件成立)，则 DS18B20 内部将报警标志置位。如果报警条件不成立，则不会设置报警标志位。也就是说，每进行一次测温就对这个标识进行一次更新。

总线控制器通过发出报警搜索命令(ECH)来对报警进行识别。

5. DS18B20 的供电

每个 DS18B20 都可以设置成两种供电方式，即数据总线供电方式(单线)和外部供电方式(3 线)。采取数据总线供电方式可以节省一根导线，但完成温度测量的时间较长。采取外部供电方式则多用一根导线，但测量速度较快。

单总线供电是在总线处于高电平状态时，因 DQ 与上拉电阻连接，故对器件提供了电源并给内部电容 C 充电；当总线变低电平时，再通过电容 C 为器件提供能量。这种为器件提供能量的方式称为寄生供电方式。

在使用“寄生供电方式”时，为了保证 DS18B20 有充足的电能，必须给单总线提供一个强上拉电路，这种电路如图 2-2(a)所示。在发出温度转换指令(44H)或拷贝暂存器指令(48H)之后，必须给总线实施强上拉(当强上拉状态保持时，不允许有其他动作)。

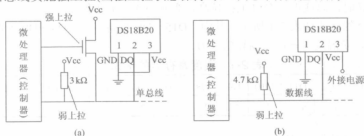


图 2-2 DS18B20 的供电方式

(a) 寄生供电方式；(b) 外部供电方式

在温度大于 100℃ 时, 不推荐使用寄生电源, 因为这时耗电较大。

外部供电方式如图 2-2(b) 所示。

6. 存储器说明

DS18B20 的存储器由一个暂存 SRAM, 一个存储高低报警触发值 TH、TL 和配置寄存器的非易失电可擦除 E²PROM 组成。其结构和配置寄存器的设置如图 2-3 所示。



图 2-3 存储器结构与配置寄存器的设置

图 2-3 中, 第 1、2 字节是只读存储器; 第 3、4 字节的 TH 和 TL 是 E²PROM 的拷贝值; 第 5 字节是配置寄存器专用来配置测量精度的; 第 6、7 和 8 字节是保留字节(禁止输入); 第 9 字节用来存储校验结果(只读)。

数据通过写暂存器指令(4EH)写入高速暂存器的 3、4 和 5 位, 数据必须以第 3 字节开始传送。为了完整地验证数据, 数据在写入后可通过指令(BEH)读出, 在读出时数据以位 1 开始从单总线传出。总线控制器传递从暂存器到 E²PROM 用拷贝指令(48H)完成, 在上电时, 数据会自动被传到暂存器中。当然, 数据也可通过召回 E²PROM 命令(B8H)从暂存器传送到 E²PROM。总线控制器在发出这条命令后, DS18B20 返回 0 表示正在召回中, 返回 1 表示完成。

7. DS18B20 的操作指令与时序

1) 操作指令

DS18B20 的操作指令(命令)如表 2-3 所示。

表 2-3 DS18B20 的操作指令

操作任务	指令(十六进制)	功能
读 ROM(Read ROM)	33H	读单只器件的编码、序列号和 CRC 码
匹配 ROM(Match ROM)	55H	命令后跟 64 位序列码以匹配在线的器件
跳过 ROM(Skip ROM)	CCH	指定跳过 64 位序列码(节省时间)
搜索 ROM(Search ROM)	F0H	搜索总线上的所有 64 位序列码
警报搜索(Alarm Search)	ECH	搜索要报警的器件

续表

操作任务	指令(十六进制)	功能
写暂存器(Write Scratchpad)	4EH	向暂存 TH 和 TL 写数据
读暂存器(Read Scratchpad)	BEH	读取暂存的内容(第 1~9 字节)
拷贝暂存器(Copy Scratchpad)	48H	将暂存的内容复制到 E ² PROM(需 10 ms 时间)
温度转换(Convert Temperature)	44H	启动一次温度转换
重新调出(Recall E ² PROM)	B8H	将 E ² PROM 中的内容拷贝到暂存器
读电源(Read Power Supply)	B4H	读出 0 表示寄生电源, 读出 1 表示外部电源

2) 工作时序

DS18B20 通过单总线端口访问 DS18B20 的步骤如下:

步骤 1: 初始化;

步骤 2: 执行 ROM 操作指令;

步骤 3: 执行 DS18B20 功能指令。

DS18B20 的每一次操作都必须满足以上三个步骤, 若缺少步骤或顺序混乱, 则器件将不会返回值。例如, 发起搜索 ROM 指令(F0H)和警报搜索指令(ECH)之后, 总线控制器必须返回步骤 1。

(1) 初始化。初始化序列包括一个由总线控制器发出的复位脉冲和其后由从机发出的存在脉冲。存在脉冲让总线控制器知道 DS18B20 在线且已做好操作准备。

初始化时序如图 2-4(a)所示。主机总线 T0 时刻发送一个复位脉冲(最短为 480 μ s 的低电平信号), 接着在 T1 时刻释放总线并进入接收状态, DS18B20 在检测到总线的上升沿之后, 等待 15~60 μ s, 然后 DS18B20 在 T2 时刻发出存在脉冲(低电平持续 60~240 μ s), 如图 2-4(a)中虚线部分所示。

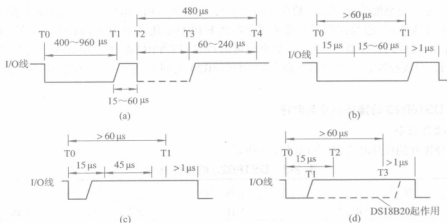


图 2-4 DS18B20 操作时序图

(a) 初始化时序; (b) 写 0 时序; (c) 写 1 时序; (d) 读时序

(2) 写时序。当主机总线在 T0 时刻从高拉至低电平时, 就产生写时间隙, 如图 2-4(b)、

(c)所示。从 T_0 时刻开始 $15\ \mu\text{s}$ 之内应将所需写的“位值”送到总线上, DS18B20 在 T_0 后 $15\sim 60\ \mu\text{s}$ 期间对总线采样。若总线是低电平, 则写入的是位“0”, 见图 2-4(b); 若总线是高电平, 则写入的是位“1”, 见图 2-4(c)。连续写 2 位间隙应大于 $1\ \mu\text{s}$ 。

(3) 读时序。主机总线在 T_0 时刻从高电平拉至低电平时, 总线只需保持低电平 $1\ \mu\text{s}$ 之后在 T_1 时刻将总线拉高产生读时间隙, 读时间隙在 T_1 时刻后 T_2 时刻前有效, T_2 距 T_0 为 $15\ \mu\text{s}$, 如图 2-4(d)所示。也就是说, T_2 时刻前主机必须完成读“位”, 并在 T_0 后的 $60\sim 120\ \mu\text{s}$ 内释放总线。

2.1.2 应用电路与编程

DS18B20 与单片机的连接如图 2-5 所示。

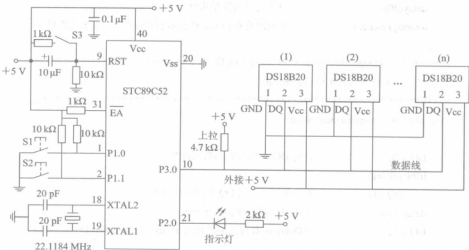


图 2-5 用 DS18B20 实现的测温电路

用 C51 实现的单点测温程序如下:

```
sbit DQ=P3^0; /*定义 DS18B20 的 I/O 口*/
/***** 延时函数 *****/
* 功能: 在 11.059 MHz 的晶振条件下调用本函数需要 24  $\mu\text{s}$ , 然后每次计数需 16  $\mu\text{s}$ ;
      在 12 MHz 的晶振下调用需 19  $\mu\text{s}$ , 每次计数需 13  $\mu\text{s}$ 。
*****/
void delay(unsigned int useconds)
{
    unsigned int s;
    for(s=0; s<useconds; s++);
}
/***** 复位函数 *****/
* 功能: 完成单总线的复位操作。
* 复位时间为 480  $\mu\text{s}$ , 因此延时时间为  $(480-24)/16 = 28.5$ , 取 29  $\mu\text{s}$ 。
```


* 经过 70 μs 之后检测存在脉冲, 因此延时时间为 $(70-24)/16=2.875$, 取 3 μs 。

```
*****
unsigned char ow_reset(void)
```

```
{
    unsigned char presence;
    DQ = 0;          /*将 DQ 线拉低*/
    delay(73);       /*保持 480  $\mu\text{s}$ */
    DQ = 1;          /*DQ 返回高电平*/
    delay(12);       /*等待存在脉冲*/
    presence = DQ;    /*获得存在信号*/
    delay(70);       /*等待时间间隔结束*/
    return(presence); /*返回存在信号, 0 表示器件存在, 1 表示无器件*/
}
```

/****** 位写入函数 ******/

* 功能: 向单总线写入 1 位值: bitval。

```
void write_bit(unsigned char bitval)
```

```
{
    DQ = 0;          /*将 DQ 拉低开始写时间间隔*/
    if(bitval==1)
        DQ = 1;      /*如果写 1, 则 DQ 返回高电平*/
    delay(15);       /*在时间间隔内保持电平值*/
    DQ = 1;          /*Delay 函数每次循环延时 16  $\mu\text{s}$ , 因此 delay(15) = 150  $\mu\text{s}$ */
}
```

/****** 字节写入函数 ******/

* 功能: 向单总线写入一个字节值 val。

```
void write_byte(unsigned char val)
```

```
{
    unsigned char i;
    unsigned char temp;
    for(i=0; i<8; i++) /*写入字节, 每次写入一位*/
    {
        temp = val>>i;
        temp &= 0x01;
        write_bit(temp);
    }
    delay(25);
}
```

/****** 位读取函数 ******/

- * 功能：从单总线上读取一位信号，所需延时时间为 15 μs ，因此无法调用前面定义的 delay() 函数，而采用一个 for() 循环来实现延时。

```
*****/ ;
unsigned char read_bit(void)
```

```
{
    unsigned char i;
    DQ = 0;          /*将 DQ 拉低开始读时间隙*/
    DQ = 1;          /*然后置高*/
    for(i=0; i<10; i++); /*延时 15  $\mu\text{s}$ */
    return(DQ);       /*返回 DQ 线上的电平值*/
}

/***** 字节读取函数 *****/
```

- * 功能：从单总线读取一个字节的值。

```
*****/
```

```
unsigned char read_byte(void)
{
    unsigned char i;
    unsigned char value = 0;
    for(i=0; i<8; i++) /*读取字节，每次读取一个字节*/
    {
        if(read_bit())
            value |= 0x01 << i; /*然后将其左移*/
        delay(120);
    }
    return(value);
}
```

```
#ifdef READ_SCRATCHPAD /*条件编译*/
```

```
/***** 读暂存器 *****/
```

- * 功能：读取暂存器中 9 个字节的数据。

```
*****/
```

```
void Read_ScratchPad(void)
```

```
{
    unsigned char j=0, pad[10];
    write_byte(0xBE);
    for(j=0; j<9; j++)
        pad[j]=read_byte();
}

#endif
```

```
/***** 读取温度函数 *****/
```

- * 功能：如果单总线节点上只有一个器件，则可以直接调用本函数。如果节点上有多个器

```

* 件, 则为了避免数据冲突, 应使用 Match ROM 函数来选中特定器件。
*****/
float Read_Temperature(void)
{
    unsigned char get[10];
    unsigned char temp_lsb,temp_msb;
    unsigned char k;
    float t;
    ow_reset(); /*复位*/
    write_byte(0xCC); /*跳过 ROM*/
    write_byte(0x44); /*启动温度转换*/
    delay(60);
    ow_reset();
    write_byte(0xCC); /*跳过 ROM*/
    write_byte(0xBE); /*读暂存器*/
    for(k=0; k<9; k++)
        get[k]=read_byte();
    temp_msb = get[1]; /*读出温度低 8 位*/
    temp_lsb = get[0]; /*读出温度高 8 位*/
    if(temp_msb >= 0x08)
    { /*为负温度*/
        temp_lsb = (~temp_lsb)+1; /*取补*/
        temp_msb &= 0x07;
        temp_msb = (~temp_msb)+1;
        t=(-1)*(temp_msb*16+temp_lsb*0.0625);
    }
    else
    {
        t=temp_lsb*0.0625+temp_msb*16;
    }
    printf( "TempC= %.3f degrees C\n", t); /*输出摄氏温度值*/
    return t;
}

```

对于多个器件, 因每一片 DS18B20 在其 ROM 中都存有唯一的 48 位序列号(在出厂前已写入片内 ROM 中), 主机在进入操作程序前必须逐一接入 DS18B20, 用读 ROM 指令(33H)将该 DS18B20 的序列号读出并登记, 当主机需要对众多在线 DS18B20 中的某一个进行操作时, 首先要发出匹配 ROM 指令(55H), 紧接着主机提供 64 位序列号的操作, 之后就是针对该 DS18B20 的操作。所谓跳过 ROM 指令, 是指之后的操作是对所有 DS18B20 而言的。

在 DS18B20 组成的测温系统中, 主机在发出跳过 ROM 指令之后, 再发出统一的温度

转换启动码 44H 就可以实现所有器件的统一转换,再经过约 1 s 后就可以用很少的时间去逐一读取多点温度值。

2.2 DS1631 温度传感器

2.2.1 硬件与功能描述

DS1631 数字温度传感器结构简单,不需要外接任何元件就能测出温度值。其最大特点是采用 I²C 总线方式,能独立输出报警信号,应用时只要对器件写入控制字,设置温度上、下限,DS1631 就能独立工作。该器件特别适合作为家用电器、办公设备、网络路由器等设备的温度保护或高温报警装置使用。

1. 主要性能特点

- (1) 测量范围为 $-55\sim+125^{\circ}\text{C}$, 其中在 $0\sim+70^{\circ}\text{C}$ 范围内具有 $+0.5^{\circ}\text{C}$ 的精度。
- (2) 温度采样分辨率为 9~12 位(包括 1 位符号位), 并可由编程决定采样位数。分辨率设置不同,转换时间也相应不同。
- (3) 电源电压范围为 $+2.7\sim+5.5\text{ V}$ 。
- (4) 采用标准的 I²C 总线接口,通过选择地址,总线上最多可挂接 8 个 DS1631 测温芯片,容易构建多点温度测量系统。
- (5) 片内含有 TH、TL 和温度寄存器。其中,TH、TL 寄存器均由两字节的 E²PROM 组成,用户通过对 TH 和 TL 寄存器的写操作,就可完成温度报警上、下限的设置。
- (6) 温度寄存器是只读存储器,由两字节的 SRAM 构成,对其进行读操作就可得到温度测量结果。

2. 内部结构与引脚说明

DS1631 的内部结构如图 2-6(a)所示,其片内的模/数转换器($\Delta\Sigma\text{ADC}$)可按预先设置的采样位数对被测温度进行实时测量,并将结果量化存入温度寄存器。

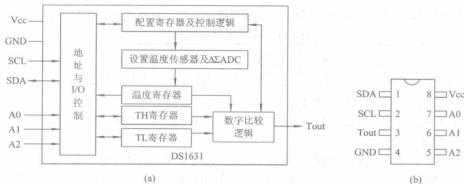


图 2-6 DS1631 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

DS1631 采用 SOP-8 脚封装,如图 2-6(b)所示。其中:

- (1) SDA 为 I²C 总线的数据输入/输出端(漏极开路, 需接 4.7 k Ω 上拉电阻)。
- (2) SCL 为 I²C 总线的时钟输入端。
- (3) Tout 为临界温度报警输出端(推挽式)。
- (4) A2~A0 为串行总线地址输入端。
- (5) Vcc 为电源脚, GND 为接地端。

3. 测量分辨率与寄存器说明

DS1631 被测温度与输出数据的关系如表 2-4 所示。12 位数据用 16 位形式表示(低 4 位为 0)。

表 2-4 DS1631 被测温度与输出数据

温度/℃	二进制输出	十六进制输出
+125	0111 1101 0000 0000	7D00H
+25.0625	0001 1001 0001 0000	1910H
+10.125	0000 1010 0010 0000	0A20H
+0.5	0000 0000 1000 0000	0080H
0	0000 0000 0000 0000	0000H
-0.5	1111 1111 1000 0000	FF80H
-10.125	1111 0101 1110 0000	F5E0H
-25.0625	1110 0110 1111 0000	E6F0H
-55	1010 1001 0000 0000	C900H

温度寄存器、TH 和 TL 由扩展的 16 位(带符号)二进制补码形式表示, 以 0.0625℃/LSB 形式递增, 其中最高位 S 为符号位。温度寄存器、TH 和 TL 的格式如表 2-5 所示。

表 2-5 温度寄存器、TH 和 TL 的格式

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
S	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻³	2 ⁻⁴	0	0	0	0

设置寄存器是 8 位读/写寄存器, 用于设置操作方式, 其格式如表 2-6 所示。

表 2-6 设置寄存器的格式

位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
DONE	THF	TLF	NVB	R1	R0	POL	1SHOT

其中, 设置寄存器各位的含义如下:

(1) DONE 位是只读位, 表示温度转换状态。“0”表示正在进行温度转换, “1”表示温度转换结束。

(2) THF 位是读/写位。“0”表示自上电后, 温度从未超过 TH 寄存器中的值; “1”表示温度曾超过上限, 并将一直保持为“1”, 直到用户对其进行写操作, 或重新上电, 或软件复位。

(3) TLF 位是读/写位。“0”表示自上电后, 温度从未低于 TL 寄存器中的值; “1”表示温度曾低于下限, 并将一直保持为“1”, 直到用户对其进行写操作, 或重新上电, 或软

件复位。

(4) NVB 位是只读位。“1”表示正在对 E^2 PROM 进行操作,“0”表示 E^2 PROM 处于空闲状态。

(5) R0 和 R1 均为读/写位,初始值为 1。R0、R1 的组合用于设置温度分辨率,具体关系如表 2-7 所示。

表 2-7 分辨率(采样位数)设置

R1	R0	温度采样位数	最大转换时间/ms
0	0	9	93.75
0	1	10	187.5
1	0	11	375
1	1	12	750

(6) POL 位是读/写位。“1”表示 Tout 高电平报警,“0”表示 Tout 低电平报警。

(7) 1SHOT 位是读/写位,用于设置温度转换模式。“1”表示单触发模式,转换命令只触发单次温度转换,之后器件进入等待状态;“0”表示连续转换模式,转换命令触发连续温度转换。

4. DS1631 接口协议

DS1631 采用标准的 I^2C 总线传输协议。在数据传输的过程中,DS1631 作为从器件,通过数据线 SDA 以及时钟线 SCL 与总线相连。开始与停止条件时序如图 2-7(a)所示。当 SCL 保持高电平时,SDA 从高电平到低电平的跳变作为数据传输的开始条件,随后主机发送控制字,包括 DS1631 的地址信息和读/写控制位,控制字的顺序是:1、0、0、1、A2、A1、A0、R/W。

根据 A2、A1 和 A0 的二进制编码,最多可允许 8 片 DS1631 挂接在同一条 I^2C 串行总线上, $R/\overline{W}=1$, 表示对 DS1631 进行读操作, $R/\overline{W}=0$, 表示进行写操作。每个字节传送结束后,接收数据一方应有一个应答信号(ACK),方可开始下一步操作。最后,在 SCL 保持高电平的情况下,SDA 从低电平到高电平的跳变作为数据传输的结束信号。

在这种操作中,首先主机发送开始条件,接着传送芯片地址信息和读/写控制位,当主机收到 DS1631 的应答信号(ACK)后,主机传送命令字,表示将要进行何种操作,其时序如图 2-7(b)所示。DS1631 在收到命令字(开始转换、读、写、结束转换)后,返回应答信号并完成相应的操作。

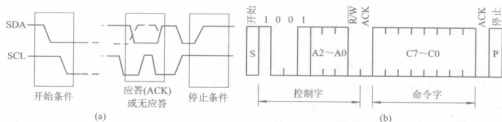


图 2-7 DS1631 时序图

(a) 启、停条件图; (b) 开始、停止转换命令时序图

1) DS1631 命令

DS1631 的操作命令如表 2-8 所示。

表 2-8 DS1631 的操作命令

操作	开始转换	停止转换	读温度寄存器	对 TH 寄存器读/写	对 TL 寄存器读/写	对配置寄存器读/写	软件复位
控制字	51H	22H	AAH	A1H	A2H	ACH	54H

2) DS1631 写操作

对于写操作, 控制字中 $R/\overline{W} = 0$ 。发送控制字后, 主机可对 DS1631 寄存器 (配置寄存器、TH 寄存器和 TL 寄存器) 进行设置。对配置寄存器和 TH 寄存器的设置时序如图 2-8 所示。如果要写 TL 寄存器, 则只要将命令字改为 A2H 即可, 主机收到应答信号后, 立即开始传送数据。对配置寄存器进行写操作时, 主机发送一字节数据。若对 TH、TL 寄存器进行写操作, 则主机必须发送两字节数据, 每收到一字节数据, DS1631 返回一个应答信号, 最后主机发送停止信号, 中止本次操作。

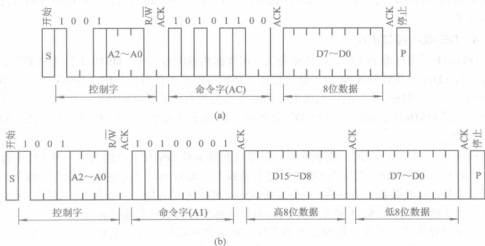


图 2-8 DS1631 “写”操作时序

(a) 写配置寄存器; (b) 写 TH 寄存器

3) DS1631 读操作

对于读操作, 控制字中 $R/\overline{W} = 1$ 。发送控制字后, 主机可对 DS1631 的相关寄存器进行读操作, 时序如图 2-9 所示。收到应答信号后, 主机应再发送一次开始信号和控制字。控制字的地址信息与第一次发送的相同, 但 R/\overline{W} (读/写) = 1, 表示将要进行读操作。DS1631 返回应答信号后, 将在下一时钟周期传送所需数据给主机。若对配置寄存器进行读操作, 则 DS1631 传送一字节数据, 主机应返回“无应答”, 然后发送停止信号, 中止本次操作, 如图 2-9(a)所示。对温度寄存器的操作时序如图 2-9(b)所示。若对 TH、TL 寄存器进行读操作, 则只要将命令字换掉即可。

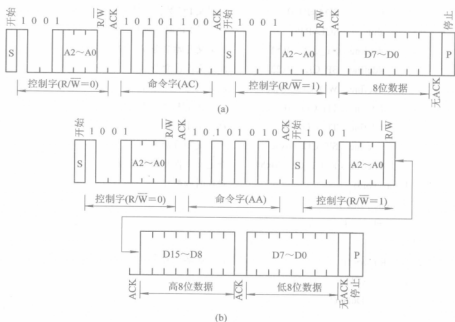


图 2-9 DS1631 “读”操作时序

(a) 读配置寄存器; (b) 读温度寄存器

2.2.2 应用电路与编程

由 DS1631 组成的多点测温电路如图 2-10 所示。其中, DS1631(1) 器件地址为 A2A1A0 = 000B, DS1631(2) 器件地址为 A2A1A0 = 001B, ..., DS1631(8) 器件地址为 A2A1A0 = 111B。

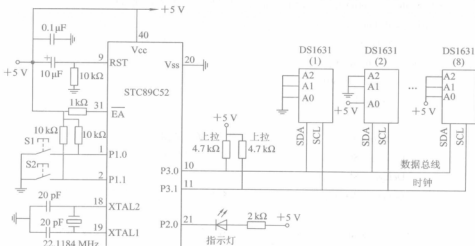


图 2-10 由 DS1631 组成的多点测温电路

按照 DS1631 的工作时序, 用 C51 实现的相关子函数如下:


```

sbit    SDA=P3^0      /*定义 I2C 数据读/写线*/
sbit    SCL=P3^1      /*定义 I2C 时钟线*/
#define  ST_CD  0x51   /*定义开始命令*/
#define  SP_CD  0x22   /*定义停止命令*/
#define  WD_CD  0xaa   /*定义温度命令*/
#define  TH_CD  0xa1   /*定义 TH 命令*/
#define  TL_CD  0xa2   /*定义 TL 命令*/
#define  ST_CD  0xac   /*定义配置命令*/
#define  RSET_CD 0x54   /*定义软件复位命令*/

void Delay(void)       /*延时函数*/
{ unsigned char j;
  for (j=0; j<4; j++);
}

void START(void)       /*开始条件*/
{ SDA=1; SCL=1; Delay();
  SDA=0; Delay();
  SCL=0;
}

void STOP(void)        /*结束条件*/
{ SDA=0; SCL=1; Delay();
  SDA=1; Delay();
  SCL=1;
}

void write_data_8(unsigned char m_data) /*写 8 位(一个字节)函数*/
{ unsigned char b,dat,ack;
  dat=m_data;          /*发送 8 位数据或控制字*/
  for(b=0; b<8; b++)
  { if(dat & 0x80)      /*高位在前*/
    SDA=1;
    else
    SDA=0;              /*先准备好数据*/
    Delay(); SCL=1;     /*延时, 产生时钟*/
    Delay(); SCL=0;
    Delay(); dat<<=1;   /*数据左移 1 位*/
  }
  SDA=1; Delay();      /*释放数据线, 延时, 准备接收 ACK*/
  SCL=1;               /*产生时钟信号*/
  if(SDA == 0)         /*读 ACK 信号*/
    ack=0;

```

```

else
    ack=1;
    SCL=0;
}

unsigned char read_data_8(unsigned char x) /*读 8bit(一个字节)函数*/
{
    unsigned char b,datai2c;
    for(b=0; b<8; b++)
    {
        datai2c<<=1;          /*数据先左移 1 位*/
        Delay(); SCL=1;       /*延时, 使 SCL 变为高电平*/
        if(SDA==0)            /*在 SCL 为高电平的情况下读 SDA*/
            datai2c&=0xfe;     /*datai2c 末位清 0*/
        else
            datai2c|=0x01;     /*datai2c 末位置 1*/
        SCL=0;                /*数据读完, 时钟线置低*/
    }
    if(x==0)                  /*x=0 时, 送 ACK 信号, 通知从机收到 1 位数据*/
    {
        SCL=0; Delay();       /*延时*/
        SDA=1; Delay();       /*准备好数据, 延时*/
        SCL=1; Delay();       /*产生时钟*/
        SCL=0; Delay();
    }
    else
        return(datai2c);
}

unsigned int read_data_16(void) /*读 16 bit(二个字节)函数*/
{
    unsigned char H_data, L_data;
    unsigned int data16;
    H_data= read_data_8(0);    /*读高 8 位数据, 有应答*/
    L_data= read_data_8(1);    /*读低 8 位数据, 无应答*/
    data16= H_data*256+ L_data; /*合成数据, 变成整型*/
    return(data16);           /*返回结果*/
}

void Write_DS1631_8(unsigned char ctr_wor,p_cd,p_data) /*写 8 位数据*/
{
    /*ctr_wor 是控制字, p_cd 是命令字, p_data 是 8 位数据*/
    START();                /*产生启动条件*/
    write_data_8(ctr_wor);   /*发送从控制字(包含器件地址)ctr_wor*/
    write_data_8(p_cd);      /*发送操作命令 p_cd*/
}

```

```

    write_data_8(p_data);          /*发送数据*/
    STOP();                        /*产生停止条件*/
}

void Write_DS1631_16(unsigned char ctr_wor, p_cd, p_data16) /*写 16 位数据*/
{
    /*ctr_wor 是控制字, p_cd 是命令字, p_data16 是 16 位数据*/
    unsigned char data8;
    START();                       /*产生启动条件*/
    write_data_8(ctr_wor);         /*发送从控制字(包含器件地址)ctr_wor*/
    write_data_8(p_cd);            /*发送操作命令 p_cd*/
    data8=p_data16/256;            /*分离出高 8 位数据*/
    write_data_8(data8);           /*发送高 8 位数据*/
    data8=p_data16%256;            /*分离出低 8 位数据*/
    write_data_8(data8);           /*发送低 8 位数据*/
    STOP();                        /*产生停止条件*/
}

unsigned char Read_DS1631_8(unsigned char ctr_wor, p_cd) /*读 8 位数据*/
{
    /*ctr_wor 是写控制字, p_cd 是命令字*/
    unsigned char rdata8;
    START();                       /*产生启动条件*/
    write_data_8(ctr_wor);         /*发送写控制字*/
    write_data_8(p_cd);            /*发送命令 p_cd*/
    START();                       /*产生启动条件*/
    write_data_8(ctr_wor|0x01);    /*发送读控制字*/
    rdata8=read_data_8(1);         /*读 8 个数据(无 ACK 信号)*/
    STOP();                        /*产生停止条件*/
}

unsigned int Read_DS1631_16(unsigned char ctr_wor, p_cd) /*读 16 位数据*/
{
    /*ctr_wor 是写控制字, p_cd 是命令字*/
    unsigned int data16;
    unsigned char Hrdata8, Lrdata8;
    START();                       /*产生启动条件*/
    write_data_8(ctr_wor);         /*发送写控制字*/
    write_data_8(p_cd);            /*发送命令 p_cd*/
    START();                       /*产生启动条件*/
    write_data_8(ctr_wor|0x01);    /*发送读控制字*/
    Hrdata8=read_data_8(0);        /*读高 8 位数据(有 ACK 信号)*/
    Lrdata8=read_data_8(1);        /*读低 8 位数据(无 ACK 信号)*/
    data16= Hrdata8*256+ Lrdata8;
    STOP();                        /*产生停止条件*/
}

```

```

return(data16);           /*返回一个整型数据*/
}

void Write_DS1631(void)    /*初始化*/
{
    START();              /*产生启动条件*/
    write_data_8(0x90);    /*发送写控制字, 90H 是指第一个温度点的测量*/
    write_data_8(ST_CD);   /*发送启动命令*/
    STOP();               /*产生停止条件*/
}

unsigned int Read_wendu(void) /*读温度(数据)*/
{ unsigned int wd_value;
  wd_value= Read_DS1631_16(0x90, WD_CD); /*读 16 位数据*/
  return wd_value;
}

```

以上作为一个例子, 只读取了地址为 000B 的器件温度。如果要读取更多的器件温度, 则只要改变器件地址就可方便实现。

2.3 DS1722 温度传感器

2.3.1 硬件与功能描述

DS1722 数字温度传感器带有 SPI/3 线接口, 无需额外元件。该器件可真正实现温度到数字的转换。通过 Motorola SPI 接口或标准的 3 线串行接口来读取温度值。温度分辨率可由用户选择, 当需要更高的温度分辨率时, 用户可以通过软件调节读数的分辨率, 范围为 8~12 位。这一点对于需要快速检测温度失控条件的系统非常有用。为了提高应用的灵活性, DS1722 具有 2.65~5.5 V 较宽的模拟电源范围。独立的数字电源允许在 1.8~5.5 V 内工作。DS1722 可被广泛应用于个人计算机、服务器、工作站、蜂窝电话、办公设备及其他热敏系统。

1. 主要性能特点

- (1) 温度测量范围: $-55\sim+120^{\circ}\text{C}$;
- (2) 温度测量精度: $\pm 2.0^{\circ}\text{C}$;
- (3) 用户可编程分辨率: 8~12 位, 相当于 $1.0\sim 0.0625^{\circ}\text{C}$ 分辨率;
- (4) 模拟电源范围: 2.65~5.5 V;
- (5) 数字电源范围: 1.8~5.5 V;
- (6) 采用 Motorola SPI 或标准的 3 线串行接口。

2. 内部结构与引脚说明

DS1722 主要由精密温度传感器、模/数转换器、SPI/3 线接口电路和数据寄存器等组成。其内部结构如图 2-11(a)所示。开始供电时, DS1722 处于能量关闭状态, 供电之后用户通过

改变寄存器分辨率使其处于连续转换模式或者单一转换模式。在连续转换模式下, DS1722 连续转换温度并将结果存于温度寄存器中, 读温度寄存器中的内容不影响其温度转换; 在单一转换模式下, DS1722 执行一次温度转换, 结果存于温度寄存器中, 然后回到关闭模式, 这种转换模式适用于对温度敏感的应用场合。在应用中, 用户可以通过程序设置分辨率寄存器来实现不同的温度分辨率的测量, 其采用位数有 8 位、9 位、10 位、11 位或 12 位五种, 对应温度分辨率分别为 1.0°C 、 0.5°C 、 0.25°C 、 0.125°C 或 0.0625°C , 温度转换结果的默认分辨率为 9 位。DS1722 有 Motorola 串行接口和标准的 3 线接口两种通信接口, 用户可以通过 SERMODE 引脚选择通信模式。

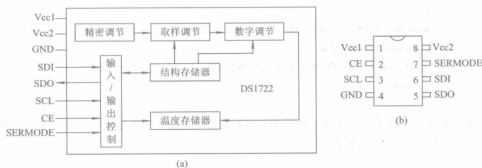


图 2-11 DS1722 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

DS1722 的引脚采用 8 引脚 SO(150 mil)、8 引脚 μMAX 封装。其中:

- (1) Vcc1 是数字电源端;
- (2) CE 为芯片使能端, 低电平有效;
- (3) SCL 是时钟输入脚, 它是数据输入/输出的同步信号;
- (4) GND 是接地端;
- (5) SDO 是数据输出脚, 在 SCL 的下降沿将数据输出;
- (6) SI 是数据输入端, 在 SCL 的下降沿将数据输入;
- (7) SERMODE 是串口模式控制端;
- (8) Vcc2 是模拟电源脚, 一般接 3V 或 5V 。

3. 温度操作方法

DS1722 传感器将温度转换成数字量后以二进制的补码格式存储于温度寄存器中, 通过 SPI 或者 3 线接口将温度寄存器地址 01H 和 02H 中的数据读出。输出数据的地址如表 2-9 所示, 数据输出的二进制与十六进制形式如表 2-10 所示。在表 2-10 中, 假定 DS1722 配置为 12 位分辨率, 数据通过数字接口连续传送, MSB(最高有效位)首先通过 SPI 传输, LSB(最低有效位)首先通过 3 线传输。

表 2-9 温度与地址

温度数据/ $^{\circ}\text{C}$								地址位
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	02H
2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0	0	0	01H

表 2-10 温度与数据输出形式

温度/℃	二进制输出	十六进制输出
+120	0111 1000 0000 0000	7800H
+25.0625	0001 1001 0001 0000	1910H
+10.125	0000 1010 0010 0000	0A20H
+0.5	0000 0000 1000 0000	0080H
0	0000 0000 0000 0000	0000H
-0.5	1111 1111 1000 0000	FF80H
-10.125	1111 0101 1110 0000	F5E0H
-25.0625	1110 0110 1111 0000	E6F0H
-55	1010 1001 0000 0000	C900H

4. 工作过程

DS1722 在 SPI 接口或 3 线接口通信方式下完成对所有寄存器的操作。其中, 状态寄存器的 D7~D5 位是“1”, D4(1SHOT)位是单步温度转换位, D3~D1(R2~R0)位是分辨率设置位(见表 2-11), D0 位(SD)是关闭断路位。在状态寄存器中, 如果 SD 位为“1”, 则不进行连续温度转换, 1SHOT 位写入“1”时, DS1722 执行一次温度转换并且把结果存在温度寄存器的地址位 01H(LSB)和 02H(MSB)中, 完成温度转换后 1SHOT 自动清“0”。如果 SD 位是“0”, 则进入连续转换模式, DS1722 将连续执行温度转换并且将全部的结果存入温度寄存器中。虽然写到 1SHOT 位的数据被忽略, 但是用户还可对该位有读/写访问权限。如果把 SD 改为“1”, 则进行中的转换将继续进行, 直至完成并且存储结果, 然后器件将进入低功耗模式。传感器上电时, 默认 1SHOT 位为“0”。R0、R1、R2 为温度分辨率位(x=任意值)。用户可以访问 R2、R1 和 R0 位, 上电时默认 R2=“0”, R1=“0”, R0=“1”(9 位转换)。此时, 通信口保持有效, 用户对 SD 位有读/写访问权限, 并且其默认值是“1”(关闭模式)。

表 2-11 温度分辨率的设置

R2	R1	R0	温度采样位数(分辨率)位	最大转换时间/s
0	0	0	8	0.075
0	0	1	9	0.1
0	1	0	10	0.3
0	1	1	11	0.6
1	x	x	12	1.2

2.3.2 应用电路与编程

由 DS1722 构成的测温电路如图 2-12 所示。电路中除 DS1722 温度传感器外, 还有 LCD 液晶显示器等电路。

在校准温度的精度时, 可以将传感器 DS1722 放入 0℃冰水混合的容器中, 逐渐给容器加热升温, 用温度计观察容器中的水温变化, 并记录传感器经单片机 STC89C52 处理后的

输出值, 根据实测结果可以用软件对误差作适当处理。

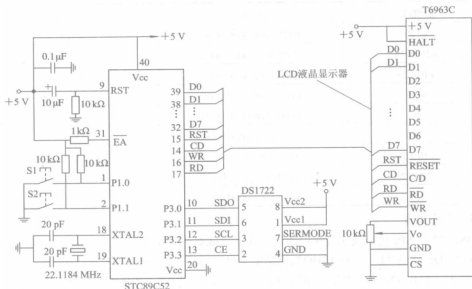


图 2-12 典型的测温应用电路

用 C51 实现的相关源程序如下:

```
#include <reg52.h>
#include <LCD12864.h> /*LCD12864 的头文件*/
#include <intrins.h> /*包含 _nop_() 的头文件*/
/*定义寄存器地址*/
#define CONFIG_WRITE 0x80 /*DS1722 配置寄存器写地址*/
#define CONFIG_READ 0x00 /*DS1722 配置寄存器读地址*/
/*配置寄存器的 8 位分别为 1 1 1 1SHOT R2 R1 R0 SD*/
#define T_LSB 0x01 /*输出温度的低 8 位*/
#define T_MSB 0x02 /*输出温度的高 8 位*/
/*配置管脚*/
sbit SDO = P3^0; /*数据输出*/
sbit SDI = P3^1; /*数据输入*/
sbit SCLK = P3^2; /*时钟*/
sbit CE = P3^3; /*使能端*/
unsigned char TEMP_L; /*全局变量, 温度的低 8 位(即小数部分)*/
unsigned char TEMP_H; /*温度的高 8 位(整数部分)*/
float temperature=0; /*浮点型的温度值*/
#define delay() { _nop_(); _nop_(); } /*延时函数*/
/*SPI 写一字节函数, 参数为要写的字节*/
void DS1722_writebyte(unsigned char sdata)
```

```

{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        SCLK=0; delay();
        SDI=(bit)(sedata & 0x80);
        sedata<<=1;
        SCLK=1; delay();
    }
}

/*SPI 读一字节函数*/
unsigned char DS1722_readbyte(void)
{
    unsigned char redata=0,i;
    for(i=0; i<8; i++)
    {
        SCLK=0; delay();
        redata<<=1;
        if(SDO==1)
            redata++; /*可用 redata=redata|0x01 替换*/
        SCLK=1; delay();
    }
    return(redata);
}

/*写配置寄存器命令，参数为命令*/
void DS1722_wrsr(unsigned char cmd)
{
    CE=0; delay(); /*片选=0*/
    DS1722_writebyte(CONFIG_WRITE); /*输入配置寄存器地址*/
    DS1722_writebyte(cmd); /*发送命令*/
    delay(); CE=1; /*释放*/
}

/*读寄存器命令，参数为要读的地址*/
unsigned char DS1722_rdsr(unsigned char address)
{
    unsigned char ch;
    CE=0; delay();
    DS1722_writebyte(address); /*要读的地址*/
    ch=DS1722_readbyte(); /*读数据*/
}

```



```

delay(); CE=1;
return ch;
}
/*等待温度转换完成*/
void wait_DS1722(void)
{
    while((DS1722_rdsr(CONFIG_READ) & 0x10)==0x10); /*当 1SHOT=1 时未结束*/
                                                    /*当 1SHOT=0 时已结束*/
}
void DS1722_init(void) /*DS1722 初始化*/
{
    SCLK=1;
    /*初始化 DS1722, 1SHOT=0, SD=1, 不进行连续转换。(R2, R1, R0)=100 为 12 位模式*/
    /*低功耗模式*/
    DS1722_wrsr(0xe9);
    delay(); /*短延时*/
}
/*读取 DS1722 温度函数, 返回的值放在 TEMP_H(高位)和 TEMP_L(低位)中*/
void Read_temperature(void)
{
    unsigned char ch;
    delay();
    ch = DS1722_rdsr(CONFIG_READ);
    DS1722_wrsr(ch<0x10); /*置 1SHOT=1, 开始温度转换*/
    wait_DS1722(); /*等待一次温度转换结束*/
    TEMP_H=DS1722_rdsr(T_MSB); /*读温度的高位, 首先通过 SPI 传输*/
    TEMP_L=DS1722_rdsr(T_LSB); /*读温度的低位*/
}
/*把字符型的温度值转换为浮点型的温度值存放在 temperature 这个全局变量中*/
void Change_Temp(void)
{
    unsigned char temp_H,temp_L;
    float ch[4];
    if((TEMP_H&0x80)==0x80) /*当温度为负值时*/
    {
        temp_H=~TEMP_H; /*高位取反, 得反码*/
        if(TEMP_L!=0x00)
            temp_L=~TEMP_L+1; /*低位不为 0 时取反加 1, 得补码*/
        else temp_L=0x00;
    }

```

```

    ch[0]=(float)((temp_L&0x80)>>7)/2; /*转换小数点后面的值*/
    ch[1]=(float)((temp_L&0x40)>>6)/4;
    ch[2]=(float)((temp_L&0x20)>>5)/8;
    ch[3]=(float)((temp_L&0x10)>>4)/16;
    temperature=(-1)*(temp_H+ch[0]+ch[1]+ch[2]+ch[3]);
}
else /*温度为正数*/
{
    temp_H = TEMP_H; /*得到原码*/
    temp_L = TEMP_L; /*得到原码*/
    ch[0]=(float)((temp_L&0x80)>>7)/2; /*转换小数点后面的值*/
    ch[1]=(float)((temp_L&0x40)>>6)/4;
    ch[2]=(float)((temp_L&0x20)>>5)/8;
    ch[3]=(float)((temp_L&0x10)>>4)/16;
    temperature= temp_H + ch[0] + ch[1] + ch[2] + ch[3];
}
}

void main(void) /*主函数部分*/
{
    DS1722_init(); /*DS1722 初始化*/
    while(1)
    {
        Read_temperature(); /*读取 DS1722 温度*/
        Change_Temp(); /*温度值转换*/
        DispFloat(temperature); /*显示浮点数, 函数定义在 LCD12864.h 中*/
    }
}

```

2.4 SHT1x/SHT7x 系列温湿度传感器

2.4.1 硬件与功能描述

SHT1x/SHT7x 系列产品是一款高集成度的温湿度传感器芯片, 提供全标定的数字输出。它采用 CMOSens 技术, 确保产品具有极高的可靠性与卓越的长期稳定性。传感器包括一个电容性聚合体湿敏感元件、一个用能隙材料制成的测温元件, 并在同一芯片上与 14 位的 A/D 转换器以及串行接口电路实现无缝连接。

每个传感器芯片都在极为精确的湿度腔室中进行标定, 以镜面冷凝式湿度计为参照。校准系数以程序形式储存在 OTP 内存中, 在标定的过程中使用。

微小的体积、极低的功耗使该系列温湿度传感器成为各类应用的首选。该器件具有品质卓越、响应快、抗干扰能力强、性价比高等优点。

该器件可被广泛用于数据采集、变送器、自动化过程控制、汽车行业、楼宇控制、暖通空调和计量测试等领域。

1. 主要性能特点

- (1) 湿度测量范围: $0\sim 100\%RH$;
- (2) 温度测量范围: $-40\sim +123.8^{\circ}C$;
- (3) 湿度测量精度: $\pm 4.5\%RH$;
- (4) 温度测量精度: $\pm 0.5^{\circ}C$;
- (5) 供电电压范围: $+2.4\sim +5.5V$;
- (6) 响应时间: <4 秒;
- (7) 低功耗(典型值为 $30\mu W$);
- (8) 全标定输出, 无需标定即可互换使用;
- (9) 全量程标定, 2线数字接口, 无需额外部件;
- (10) 可完全浸没使用。

2. 内部结构与引脚说明

SHT1x/SHT7x 器件的内部结构主要由温度传感器、湿度传感器、高精度放大器、14 位 A/D 转换器、校验存储器和数字接口等电路组成, 如图 2-13(a)所示。其引脚排列见图 2-13(b)。其中:

- (1) SCL 是串行时钟输入, 用于微处理器与 SHTxx 之间的通信同步。
- (2) SDA 是数据输入/输出端。SDA 在 SCL 时钟下降沿之后改变状态, 并仅在 SCL 时钟上升沿有效。数据传输期间, 在 SCL 为高电平时, SDA 必须保持稳定。使用时, SDA 需要一个 $10k\Omega$ 的外部上拉电阻。
- (3) Vcc 是电源脚, 通常接 $+2.4\sim +5.5V$ 电压。

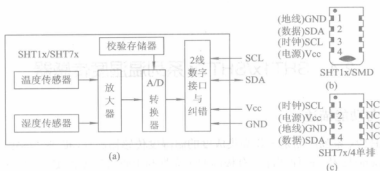


图 2-13 器件内部结构与引脚排列

(a) 内部框图; (b)、(c) 引脚排列

SHT1x/SHT7x 系列器件的具体型号如表 2-12 所示。

表 2-12 型号与精度

具体型号	湿度精度/%RH	温度精度/℃(在 25℃时)	封装形式
SHT10	±4.5	±0.5	SMD(LCC)
SHT11	±3.0	±0.4	SMD(LCC)
SHT15	±2.0	±0.3	SMD(LCC)
SHT71	±3.0	±0.4	4-PIN 单排直插
SHT75	±1.8	±0.3	4-PIN 单排直插

3. 使用说明

1) 发送命令

用一组“启动传输”时序进行数据传输的初始化过程包括: 在 SCL 时钟高电平期间 SDA 变为低电平, 紧接着 SCL 变为低电平, 随后在 SCL 为高电平时, SDA 再次翻转为高电平。后续命令包含三个地址位(目前只支持“000”)和五个命令位(00011H 是温度测量命令, 00101B 是湿度测量命令, 00111B 是读状态寄存器命令, 00110B 是写状态寄存器命令, 11110B 是软复位命令)。SHTxx 会以以下述方式表示已正确接收到指令: 在第 8 个 SCL 时钟的下降沿之后, 将 SDA 下拉为低电平(ACK 位); 在第 9 个 SCL 时钟的下降沿之后, 释放 SDA(恢复高电平)。

2) 测量时序

发送一组测量命令(地址+命令)后, 即相对湿度 RH 是“00000101B”, 温度 T 是“00000011B”, 控制器要等待测量结束。这个过程需要大约 20/80/320 ms, 分别对应 8/12/14 位测量。确切的时间随内部晶振精度最多可能有 30% 的变化。SHTxx 通过下拉 SDA 至低电平并进入空闲模式表示测量的结束。控制器再次触发 SCL 时钟前, 必须等待这个“数据备妥”信号来读出数据。检测数据可以先被存储, 这样控制器可以继续执行其他任务, 在需要时再读出数据。

然后, 传输 2 个字节的测量数据和 1 个字节的 CRC 奇偶校验。微处理器需要通过下拉 SDA 为低电平, 以确认每个字节。所有的数据从 MSB 开始(例如对于 12 位数据, 从第 5 个 SCL 时钟起算做 MSB; 对于 8 位数据, 首字节则无意义)。CRC 数据的确认位用来表明通信结束。如果不使用 CRC-8 校验, 则控制器可以在测量值 LSB 后, 通过保持确认位 ACK 高电平来中止通信。在测量和通信结束后, SHTxx 自动转入休眠模式。测量 RH 的时序如图 2-14 所示。图中, $RH = 00001001\ 00110001B = 2353 = 75.79\%$ (未进行温度补偿)。

对于温度的测量只要将命令变成 00000011 即可。

如果与 SHTxx 通信中断, 则下列时序可以复位串口, 即当 SAD 保持高电平时, 触发 SCL 时钟 9 次或更多, 在下次指令前, 再发送一个“传输启动”时序, 就可完成软复位。这些时序只复位串口, 状态寄存器内容仍然保留。

3) 状态寄存器

SHTxx 的某些高级功能可以通过状态寄存器实现。状态寄存器的“写”与“读”时序如图 2-15 所示。

状态寄存器的设置如表 2-13 所示。

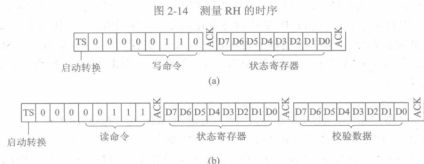
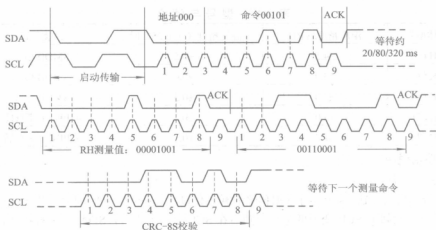


图 2-15 对状态寄存器的操作时序

(a) 写时序; (b) 读时序

表 2-13 状态寄存器的设置

位(bit)	类型	含 义	默认值
7		预留位	0
6	读	电量不足, 读数“0”表示 $V_{cc} < 2.47\text{ V}$	x 无默认值
5		预留位	0
4		预留位	0
3		仅供测试, 不使用	0
2	读/写	加热	0 关
1	读/写	不从 OTP 加载	0 加载
0	读/写	“1”=8 bit RH/12 bit T, “0”=12 bit RH/14 bit T	0

从 2-13 表中可以看出:

- (1) 默认的测量分辨率分别为 14 bit(温度)、12 bit(湿度), 也可分别降至 12 bit 和 8 bit。通常在高速或超低功耗的应用中采用该功能。
- (2) “电量不足”功能可监测到 V_{cc} 电压低于 2.47 V 的状态。精度为 $\pm 0.05\text{ V}$ 。
- (3) 芯片上集成了一个可通断的加热元件。接通后, 可将 SHTxx 的温度提高大约 5~

15℃, 功耗增加约 8 mA(在 5 V 时)。该芯片可应用于:

① 比较加热前后的温度和湿度值, 可以综合验证两个传感器元件的性能。

② 在高湿度(>95%RH)环境中, 加热传感器可防止凝露, 同时缩短其响应时间, 提高测量精度。

注意, 加热后较之加热前, SHTxx 的显示温度值略有升高, 相对湿度值稍有降低。

4) 物理信号的转换

(1) 相对湿度。为了补偿湿度传感器的非线性以获取准确数据, 建议使用如下公式修正输出数值:

$$RH = C_1 + C_2 \cdot SO_{RH} + C_3 \cdot SO_{RH}^2$$

上式中的各系数见表 2-14。

表 2-14 相对湿度转换系数

SO_{RH}	C_1	C_2	C_3
12 位	-4	0.0405	-2.8×10^{-6}
8 位	-4	0.648	-7.2×10^{-4}

对高于 99%RH 的那些测量值, 表示空气已经完全饱和, 必须被处理成显示值均为 100%RH。

(2) 温度。由能隙材料 PTAT(正比于绝对温度)研发的温度传感器具有极好的线性。可用如下公式将数字输出转换为温度值:

$$\text{温度} = d_1 + d_2 \cdot SO_T$$

上式中的各系数如表 2-15 所示。

表 2-15 温度转换系数

Vcc/V	d_1	d_1	SO_T/bit	d_2	d_2
+5	-40.00	-40.00	14	0.01	0.018
+4	-39.75	-39.55	12	0.04	0.072
+3.5	-39.66	-39.39	14	0.01	0.018
+3	-39.60	-39.28	12	0.04	0.072
+2.5	-39.55	-39.19	14	0.01	0.018

2.4.2 应用电路与编程

SHT1x 系列温湿度传感器与 STC89C52 单片机应用电路如图 2-16 所示。SHT1x 的供电电压为+5 V。传感器上电后, 要等待 11 ms 以越过“休眠”状态。在此期间不能发送任何指令。

SHT1x 虽然采用的是串行接口, 但与 I²C 接口不兼容。编程时要根据图 2-14 和图 2-15 的相关时序进行。

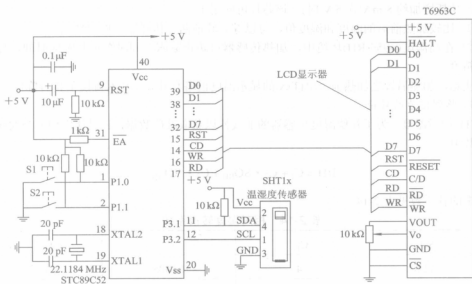


图 2-16 SHT1x 传感器的典型应用

下面是结合图 2-16 的应用电路，通过 C51 所实现的相关温度、湿度采集函数。

```

/*定义引脚*/
sbit SHT_DATA=P3^1;          /*数据端 SDA*/
sbit SHT_SCK=P3^2;           /*时钟端 SCL*/
void delay(void)              /*延时函数*/
{ unsigned char i;
  for (i=0; i<10; i++);
}

typedef union
{ unsigned int i;
  float f;
}value;

enum {TEMP,HUMI};             /*温度、湿度选择量*/

#define noACK 0                /*继续传输数据，用于判断是否结束通信*/
#define ACK 1                  /*结束数据传输*/

/*地址 命令 读/写*/
#define STATUS_REG_W 0x06      /*000 0011 0 */
#define STATUS_REG_R 0x07      /*000 0011 1 */
#define MEASURE_TEMP 0x03      /*000 0001 1 */
#define MEASURE_HUMI 0x05      /*000 0010 1 */
#define RESET 0x1e             /*000 1111 0 */
/*-----*/

```

```

unsigned char s_write_byte(unsigned char value)
/*写一个字节, 并检查 ACK*/
{
    unsigned char i,error=0;
    for (i=0x80; i>0; i/=2)          /*高位为 1, 循环右移*/
    { if (i & value) SHT_DATA=1;      /*和要发送的数相与, 结果为发送的位*/
      else SHT_DATA=0;
      SHT_SCK=1;
      delay();
      SHT_SCK=0;
    }
    SHT_DATA=1;                      /*释放数据线*/
    SHT_SCK=1;                      /*用于得到 ACK 的时钟*/
    error=SHT_DATA;                 /*检查 ACK(SHT_DATA 将下拉)*/
    SHT_SCK=0;
    return error;                   /*error=1, 通信错误*/
}

unsigned char s_read_byte(unsigned char ack)
/*读一个字节并具有 ACK 信号*/
{
    unsigned char i,val=0;
    SHT_DATA=1;                    /*释放数据线*/
    for (i=0x80; i>0; i/=2)        /*高位为 1, 循环右移*/
    {
        SHT_SCK=1;
        if (SHT_DATA) val=(val | i); /*读数据位*/
        SHT_SCK=0;
    }
    SHT_DATA=!ack;                 /*如果是校验, 则读取完后结束通信*/
    SHT_SCK=1;
    delay();
    SHT_SCK=0;
    SHT_DATA=1;                   /*释放数据线*/
    return val;
}

/*-----启动传送条件-----*/
void s_transstart(void)           /*启动传送条件*/
{
    SHT_DATA=1; SHT_SCK=0;        /*初始化数据线和时钟线*/
    delay(); SHT_SCK=1;
}

```



```

delay(); SHT_DATA=0;
delay(); SHT_SCK=0;
delay(); SHT_SCK=1;
delay(); SHT_DATA=1;
delay(); SHT_SCK=0;
}
/*-----产生复位-----*/
void s_connectionreset(void) /*产生复位*/
{
    unsigned char i;
    SHT_DATA=1; SHT_SCK=0; /*SHT_DATA 保持高位*/
    /*SHT_SCK 时钟触发 9 次, 发送启动传输, 通信即复位*/
    for(i=0; i<9; i++)
    {
        SHT_SCK=1;
        SHT_SCK=0;
    }
    s_transstart(); /*启动传送条件*/
}
/*-----软件复位-----*/
unsigned char s_softreset(void) /*软件复位*/
{
    unsigned char error=0;
    s_connectionreset(); /*产生复位*/
    error+=s_write_byte(RESET); /*发送复位命令*/
    return error; /*error=1, 通信错误*/
}
unsigned char s_read_statusreg(unsigned char *p_value, unsigned char *p_checksum)
/*读状态寄存器的值(8 bit)*/
{
    unsigned char error=0;
    s_transstart(); /*启动传送条件*/
    error=s_write_byte(STATUS_REG_R); /*发送命令*/
    *p_value=s_read_byte(ACK); /*读状态寄存器的值*/
    *p_checksum=s_read_byte(noACK); /*读校验码*/
    return error; /*error=1, 通信错误*/
}
unsigned char s_write_statusreg(unsigned char *p_value)
/*写状态寄存器(8 bit)*/
{

```

```

    unsigned char error=0;

    s_transstart();                                /*启动传送条件*/
    error+=s_write_byte(STATUS_REG_W);            /*发送命令*/
    error+=s_write_byte(*p_value);                /*发送状态寄存器的值*/
    return error;                                  /*error>=1, 通信错误*/
}

/*-----相关函数说明-----*/

unsigned char s_measure( *p_value, *p_checksum, mode)
unsigned char *p_value, unsigned char *p_checksum, unsigned char mode
/*进行温度或者湿度转换, 由参数 mode 决定转换内容*/
{
    unsigned error=0;
    unsigned int i;
    s_transstart();
    switch(mode){                                  /*发送命令*/
        case TEMP : error+=s_write_byte(MEASURE_TEMP); break;
        case HUMI : error+=s_write_byte(MEASURE_HUMI); break;
        default : break;
    }
    for (i=0; i<65535; i++)
        if(SHT_DATA==0) break;                    /*等待测量结束*/
    if(SHT_DATA)
        error+=1;                                  /*如果长时间数据线没有拉低, 则说明测量错误*/
    *(p_value) =s_read_byte(ACK);                  /*读第一个字节, 高字节(MSB)*/
    *(p_value+1)=s_read_byte(ACK);                 /*读第二个字节, 低字节(LSB)*/
    *p_checksum =s_read_byte(noACK);              /*读 CRC 校验码*/
    return error;
}

/*-----计算温度和湿度-----*/
void calc_sht11(float *p_humidity, float *p_temperature)
{
    /*补偿及输出温度和相对湿度 */
    const float C1=-4.0;                          /*for 12 bit 湿度修正公式*/
    const float C2=+0.0405;                       /*for 12 bit 湿度修正公式*/
    const float C3=-0.0000028;                    /*for 12 bit 湿度修正公式*/
    const float T1=+0.01;                         /*for 14 bit @ 5 V 温度修正公式*/
    const float T2=+0.00008;                      /*for 14 bit @ 5 V 温度修正公式*/
    float rh=*p_humidity;
    float t=*p_temperature;
    float rh_lin;
    float rh_true;

```

```

float t_C;
t_C=(t*0.01 - 40);          /*补偿温度*/
rh_lin=C3*rh*rh + C2*rh + C1; /*相对湿度非线性补偿*/
rh_true=(t_C-25)*(T1+T2*rh)+rh_lin; /*相对湿度对于温度的依赖性补偿*/
if(rh_true>100)rh_true=100;    /*湿度最大修正*/
if(rh_true<0.1)rh_true=0.1;   /*湿度最小修正*/
*p_temperature=t_C;           /*返回温度结果*/
*p_humidity=rh_true;          /*返回湿度结果*/
}
/*-----主函数-----*/
void main()                   /*作为一个例子，将结果输出到显示器上*/
{
    value humi_val,temp_val;
    unsigned char error,checksum;
    unsigned int i;
    Init_com9600();            /*串口初始化*/
    s_connectionreset();        /*产生复位*/
    while(1)
    { error=0;
      error+=s_measure((unsigned char*)&humi_val,i,&checksum,HUMI); /*测量*/
      error+=s_measure((unsigned char*)&temp_val,i,&checksum,TEMP); /*测量*/
      if(error!=0)
          s_connectionreset(); /*如果错误，则产生复位*/
      else
      { humi_val.f=(float)humi_val.i; /*强制转换为浮点数*/
        temp_val.f=(float)temp_val.i; /*强制转换为浮点数*/
        calc_sth11(&humi_val.f,&temp_val.f); /*计算温度和湿度*/
        printf("temp:%5.1fC humi:%5.1f%%\n",temp_val.f,humi_val.f);
      }
      for (i=0; i<60000; i++); /*长延时*/
    }
}

```

2.5 MEMSIC 加速度传感器

2.5.1 硬件与功能描述

MEMSIC 加速度传感器是基于单片 CMOS 集成电路工艺制造出来的一个完整的双轴加速度测量系统。MEMSIC 器件是以可移动的热对流小气团作为重力块，类同于其他加速度

传感器的重力模块。该器件通过测量由加速度引起的内部温度的变化来测量加速度。MEMSIC 传感器中的质量块是气体状态的质量块,同传统的实体质量块相比具有很大的优势。MEMSIC 器件不存在电容式传感器所存在的粘连、颗粒等问题,同时能抵抗 50 000 g 的冲击,这使得 MEMSIC 器件的次品率和故障率很低。

1. 主要性能特点

- (1) 具有 2 轴(X、Y)加速度信号输出;
- (2) 每一轴与水平的倾斜角: $0^{\circ} \sim 90^{\circ}$;
- (3) 测量范围: $\pm 1 \sim \pm 100 \text{ g}$;
- (4) 灵敏度在整个温度范围($-40 \sim +105^{\circ}\text{C}$)内的变化小于 3%;
- (5) 内部数字电源 Vcc 的输入范围: $+3 \sim +5.25 \text{ V}$;
- (6) 内部模拟电源 Vda 的输入范围: $+3 \sim +5.25 \text{ V}$;
- (7) 参考电源输出: $+2.5 \text{ V}$, 输出电流: $100 \mu\text{A}$;
- (8) 加速度最大输出电流: $<100 \mu\text{A}$ 。

2. 内部结构与引脚说明

MEMSIC 加速度传感器的内部结构如图 2-17(a)所示。一个被放置在硅芯片中央的热源在一个空腔中产生一个悬浮的热气团(图中的 R)。同时由铝和多晶硅组成的热电偶组(图中的 L)被等距离地放置在热源的四个方向。在未受到加速度或水平放置时,温度的下降梯度是以热源为中心完全对称的,此时所有四个热电偶组因感应温度而产生的电压是相同的。由于自由对流热场的传递性,任何方向的加速度都会扰乱热场的轮廓,从而导致其不对称,此时四个热电偶组的输出电压会出现差异。热电偶组输出电压的差异是直接与其所感应的加速度成比例的。在加速度传感器内部有两条完全相同的加速度信号传输路径:一条用于测量 X 轴上所感应的加速度,另一条用于测量 Y 轴上所感应的加速度。

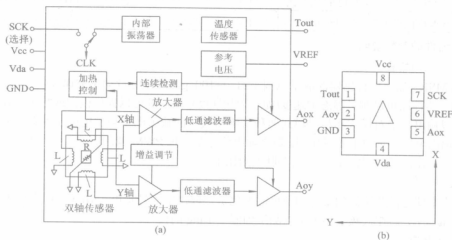


图 2-17 内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

MEMSIC 加速度传感器的引脚排列如图 2-17(b)所示。封装采用低高度 LCC 表面贴形式(封装尺寸为 $5\text{ mm} \times 5\text{ mm} \times 2\text{ mm}$)。它还包含一个内置的温度传感器和参考电压输出。其中:

- (1) Tout 是温度模拟电压输出;
- (2) Aoy 是 Y 轴加速度信号输出;
- (3) GND 是参考地;
- (4) Vda 是模拟电源输入脚, 一般接 5 V ;
- (5) Aox 是 X 轴加速度信号输出;
- (6) VREF 是 2.5 V 参考电源输出;
- (7) SCK 是可选外部时钟输入脚(通常接地);
- (8) Vcc 是数字电源输入脚, 一般接 5 V 。

MEMSIC 加速度传感器的型号选择如表 2-16 所示。

表 2-16 MEMSIC 系列产品的型号选择

型 号	范围/g	灵敏度	分辨率/mg	电源/V	轴数	输出模式	噪声
MXR2010AL	35	10 mV/g	2.0	5	2	模拟	1.25
MXA2050AL	10	50 mV/g	2.0	5	2	模拟	0.75
MXA2300JV	1.5	300 mV/g	2.0	3	2	模拟	1
MXD2020EL	1	20% 占空比	1.0	5	2	PWM	0.2
MXA2500EL	1	500 mV/g	1.0	5	2	模拟	0.2
MXA6500MP	1	500 mV/g	1.0	3	2	模拟	0.4
MXR6500MP	1.7	500 mV/g	1.0	3	2	模拟	0.4
MXR6500EP	0.5	500 mV/g	1.0	3	2	模拟	0.4
MXD6125MP	2	12.5% 占空比	1.0	3	2	PWM	0.4
MXD6150MP	5	150 mV/g	1.0	3	2	模拟	0.4

3. 器件的参数说明

1) 量程和输出

MEMSIC 加速度传感器的测量范围是 $\pm 1 \sim \pm 100\text{ g}$ 。除了测量动态加速度(如振动)外, MEMSIC 器件还可以测量静态加速度(如重力加速度)。该器件可以提供模拟或数字的输出信号。模拟输出有绝对模式和相对模式两种。绝对模式的输出电压与供电电压无关, 而相对模式的输出电压和供电电压成比例。数字输出信号是一种 PWM 调制后的、和加速度大小成正比的占空比信号(高电平占一个周期脉宽的比率)。

2) 分辨率

分辨率就是能测量到的最小加速度的变化量, 取决于信号噪声。MEMSIC 的典型噪声低于 $1\text{ mg}/\sqrt{\text{Hz}}$ 。在低频条件下可以测量到低于 1 mg 的信号。

3) 频率响应

频率响应就是对快速变化的加速度的反应能力, 由结构来决定。对 MEMSIC 器件来说, 在 -3 dB 处频响为 30 Hz 。通过外部扩展, 频响可以扩展到 160 Hz 以上。

4) Aox(X轴)加速度输出信号

与 X 轴加速度输出信号相连的器件的输入阻抗必须足够高以保证此脚的输出电流不大于 $100\ \mu\text{A}$, 其灵敏度在出厂前被设置成与 Y 轴相同, 但可以根据用户的要求将两个轴的灵敏度设置成不同的值(详细情况请与 MEMSIC 厂商联系)。

5) Aoy(Y轴)加速度输出信号

与 Y 轴加速度输出信号相连的器件的输入阻抗必须足够高以保证此脚的输出电流不大于 $100\ \mu\text{A}$, 灵敏度在出厂前被设置成与 X 轴相同, 但可以根据用户的要求将两个轴的灵敏度设置成不同的值。

6) Tout 温度输出信号

内部温度传感器的输出信号是模拟电压, 反映的是管芯衬底的温度。此电压可用于测量周围环境温度的变化量(而不是对温度直接测量), 当环境温度发生变化时, Tout 的输出电压是相对于 25°C 时的电压差值。用此差值可以对传感器的零点偏置和灵敏度进行补偿。

7) SCK 信号

MEMSIC 标准产品选择的是内部时钟($800\ \text{kHz}$), 当选择内部时钟时, 此脚必须接地。根据客户的特殊要求, MEMSIC 可以定制使用外部时钟的产品。外部时钟的频率范围为 $400\ \text{kHz}\sim 1.6\ \text{MHz}$ 。

4. 信号的补偿

1) 灵敏度的补偿

所有热电偶式加速度传感器的灵敏度都会随温度而变化。这种灵敏度的变化完全由热传导的物理特性所决定。制造过程中的个体差异不会影响这种灵敏度的变化, 所以这种灵敏度的变化不存在个体之间的不同。灵敏度随温度的变化遵循以下公式:

$$S_i \times T_i^{2.67} = S_f \times T_f^{2.67}$$

其中, S_i 是在任何初始温度 T_i (如 25°C) 时的灵敏度, 而 S_f 是在任何最终温度 T_f 时的灵敏度。温度单位均为绝对温度单位 F。温度的幂值系数 T 对于不同类型的器件会有一些偏差(比如, EL 型的 T 是 2.90, 而 AL 型的是 2.67)。

对于游戏应用领域, 游戏机或操纵器是在室温下工作的, 故环境温度可以认为基本不变, 因此就没有必要对器件的灵敏度进行补偿, 可以通过游戏前的校准或通过玩家的直觉来弥补。

对于那些允许灵敏度有百分之几变化的应用领域, 上述公式可以用一个线性函数来近似。用这种近似的方法(通过一个有 $-0.9\%/^\circ\text{C}$ 增益的外部电路)可以将灵敏度的变化限制在 2.6% 以内(以室温时的灵敏度为基准, 温度从 0°C 变化到 $+50^\circ\text{C}$)。

对于性能要求比较高的应用场合, 可以用一个低价位的 MCU 来完成以上公式的计算。

2) 零点温漂的补偿

同所有其他的加速度测量技术一样, 每个 MEMSIC 器件都有一个特定的零点温漂特性。每个应用方案可接受的零点温漂值各不相同。标准的 MEMSIC 器件的温漂系数是 $\pm 2\ \text{mg}/^\circ\text{C}$, 新型的低噪声器件的温漂系数小于 $\pm 1\ \text{mg}/^\circ\text{C}$ 。

对于高精度应用项目, 当零点温漂值的精度不能满足要求时, 可以逐个测定其温漂系

数, 再进行补偿。进行补偿需要得到每个器件的温漂系数, 因为每个器件具有不同的温漂特性。补偿时, 一个和温漂相反极性的偏移量被加到了加速度输出信号中。图 2-18 就是一个使用模拟电路进行线性补偿的例子。在这个电路中, 加速度传感器的输出被加上或者减去了温度补偿偏移量。

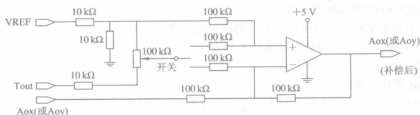


图 2-18 模拟电路补偿温漂的方法

补偿的过程如下：在室温下启动时将 $100\text{ k}\Omega$ 电位器的滑片置于 VREF 端。接着在所要求的温度下观察器件的温度漂移方向，把开关打到运放的反相输入端，调整电位器，使零点温漂值和室温下的值相同。

运用类似的概念，各种数字补偿技术也可以实现。数字技术可以提供更好的补偿，因为它可以进行非线性的零点温漂补偿。单片机或微控制器可用来处理数据。像用模拟电路进行补偿一样，首先需测出零漂的特性曲线。这样做是为了产生一张特性近似表或一个数学近似表达式。例如，这种温漂可以近似地用以下公式表示：

$$\text{温漂值} = aT^2 + bT + c$$

其中，a、b、c 是每个器件所特有的常数；T 是被测温度。

通常，经过处理后的输出值为

$$\text{补偿后的输出值} = \text{加速度} - \text{温漂值}$$

5. 倾斜测量及最高分辨率

MEMSIC 加速度传感器的一个最普遍的应用是用于倾角的测量。加速度传感器通过感知地球重力加速度在其测量轴上的分量的大小来确定物体的倾斜角度。当 MEMSIC 加速度传感器被水平放置(两个灵敏轴与水平面平行)时，它对位置或倾角的变化最为敏感。当它被垂直放置(两个灵敏轴与水平面垂直)时，对于位置或倾角变化的敏感度将下降。表 2-17 描述了器件从 $+90^\circ$ 到 0° 倾斜过程中 X 轴和 Y 轴输出值的相应变化。请注意当一根轴(每倾斜 1°)的输出变化较小时，另一根轴的输出变化则较大。把两个轴的这一变化特性相互弥补可以设计成一款价位低、精度较高的倾角仪。

表 2-17 倾斜过程中 X 轴和 Y 轴的变化

X 轴与水平夹角 $\theta(^{\circ})$	X 轴		Y 轴	
	X 轴输出	每倾斜 1° 的变化/mg	Y 轴输出	每倾斜 1° 的变化/mg
90	1.00	0.15	0.000	17.45
85	0.996	1.37	0.087	17.37
80	0.985	2.88	0.174	17.16
70	0.940	5.86	0.342	16.35

续表

X 轴与水平夹角 / (°)	X 轴		Y 轴	
	X 轴输出	每倾斜 1° 的变化/mg	Y 轴输出	每倾斜 1° 的变化/mg
60	0.866	8.59	0.500	15.04
45	0.707	12.23	0.707	12.23
30	0.500	15.04	0.866	8.59
20	0.342	16.35	0.940	5.86
10	0.174	17.16	0.985	2.88
5	0.087	17.37	0.996	1.37
0	0.000	17.45	1.000	0.15

加速度传感器的分辨率受其噪声的限制。输出噪声的大小随频带宽度而变化。用一个外部低通滤波器可以降低噪声,提高信噪比和分辨率。输出噪声以测量频宽的平方根为基本刻度。噪声的峰-峰值决定了分辨率的大小。噪声的峰-峰值近似地等于其均方根值的 6.6 倍(包括 0.1% 的平均不确定因素)。对于一个简单的低通滤波器,它的均方根噪声可以通过公式计算出来:

$$\text{噪声(mg rms)} = \text{噪声}(1 \text{ mg}/\sqrt{\text{Hz}}) \times \sqrt{\text{带宽} \times 1.6}$$

6. 外部滤波器的设计

对于那些只测量动态加速度(如振动)的应用领域,建议对传感器的输出信号进行交流耦合,如图 2-19(a)所示。交流耦合的优点在于它可以排除零点偏置的个体差异以及零点偏置的温漂问题。该图是一个典型的高通滤波器(HP)电路,它在 -3 dB 处的频响为 $f = 1/(2\pi RC)$ 。在许多应用中人们希望将 -3 dB 处的频响设置得非常低,以便能检测到一些低频加速度。为了达到这一目的,设计人员有时会采用不合理的大电容,在此建议采用软件高通滤波的方法来实现。

对于频响较低的应用领域,如倾角仪或水平尺,可以外加低通滤波器来抑制噪声,提高分辨率。图 2-19(b)是一个典型低通滤波器(LP)电路,它在 -3 dB 处的频响为 $f = 1/(2\pi RC)$ 。

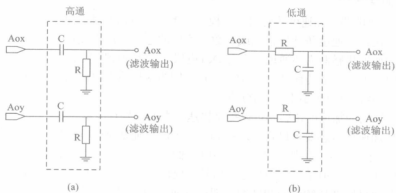


图 2-19 滤波器的设计电路

7. 器件的使用

1) 低功耗的设计方法

在电源功耗受限制的应用领域,采用脉冲供电方式可以延长电池的使用寿命。必须注意的是当采用脉冲供电时,器件的测量频宽将受到其开机时间的限制。例如,器件在 3 V 供电时的开机时间是 40 ms。如采用 40 ms 开、40 ms 关的脉冲供电方式,那么其周期为 40 ms(频率为 12.5 Hz),也就是说频率为 6.25 Hz 的加速度变化可以被检测到。

由于倾角的变化比较缓慢,因此在倾角测量的应用领域中可以有效地采用脉冲供电方式。

2) 频率响应的扩展

热电偶式加速度传感器的频响是由其内部所用气体的物理特性、热对流原理以及传感器电子学所决定的。由于 MEMSIC 在大批量生产中所使用的气体的特性完全一致,因此可以用一个简单的电路来对所有器件进行相同的补偿。对于大多数应用来说,不需要因为器件的不同而对补偿电路进行调整。

一个简单的补偿网络由两个运算放大器以及一些用于提高增益、提升频率的电阻和电容组成,如图 2-20 所示。此电路中所用器件为模拟输出,电源电压是 +5 V,放大倍数是 2,所以放大后的零点偏置为 2.5 V,同时灵敏度为原先的两倍。与 1.5 μF 、0.01 μF 电容并联的 14.3 k Ω 及 5.9 k Ω 电阻可对网络的增益进行微调,以对高频衰减进行补偿。其他电阻、电容用于降低噪声。

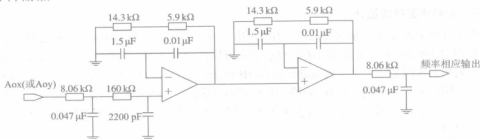


图 2-20 频率响应的扩展电路

3) 器件的接口

MEMSIC 热电偶式加速度传感器具有模拟或者数字输出两种。每个加速度传感器同时也提供温度输出。加速度传感器模拟或者数字输出的选择取决于硬件和软件的情况。具有模拟输出的加速度传感器需要通过一个 ADC(模拟/数字转换器)连接到处理器(或单片机)。在这个应用中,需要一个具有串行接口的 ADC,因为采样速率很低(小于 5 Hz)。ADC 的分辨率必须大于 10 位以达到小于 1° 的分辨率。

选择模拟输出的加速度传感器可以提供快速易得的信号且成本较低。

数字输出加速度传感器可以直接连接到处理器的普通数字输入引脚上,但是需要更多的处理周期来处理信号。虽然加速度传感器内部具有模拟/数字信号转换器,输出信号是 PWM(脉宽调制)形式,但是要得到加速度信号,处理器还要进行 PWM 信号占空比的计算。

占空比是通过脉宽持续时间和脉冲周期计算而来的。测量脉宽持续时间和脉冲周期需要额外的计时器或者使用内部的时钟进行周期计数。

选择数字输出的加速度传感器可以省去接口的费用,不过会增加程序的运算量。

4) 信号处理

从加速度传感器得到的信号是作用在加速度传感器感应轴上的重力的表现。进一步的信号处理可以提高精度和得到倾斜角度。

信号处理的第一步是降低噪声。MEMSIC 加速度传感器是低噪声器件,用于倾角测量的加速度传感器其噪声量级很小。例如,在水平角度附近,每变化 1° 的倾角表现为加速度信号的变化是 17 mg 。对于灵敏度为 500 mV/g 的模拟输出加速度传感器来说,表现为 8.5 mV 的输出值变化。一般通过限制信号的带宽,可以明显减小噪声的大小。如果 LP(低通滤波器)的限制频率是 10 Hz ,那么噪声将被限制在 0.1° 之内。模拟输出加速度传感器信号的低通滤波器可以由硬件或者软件实现,而数字输出加速度传感器信号的只能由软件实现。

2.5.2 应用电路与编程

选用 MXR6500MP 加速度传感器的应用电路如图 2-21 所示。电路设计中,采用 X 轴、Y 轴温度补偿电路的典型接法,图中的元件参数同图 2-18。 R_{14} 、 C_1 、 R_{15} 和 C_2 是低通滤波器,截止频率可根据实际应用而定(一般不宜过高)。单片机采用带有 10 位 A/D 转换器的 STC12C5410AD 器件。

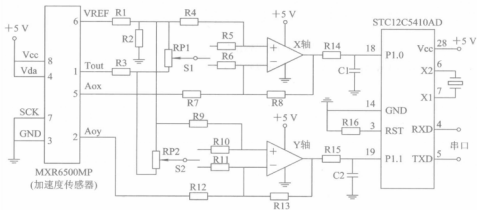


图 2-21 加速度传感器的应用电路

加速度传感器的安装位置与所测试的参数有关。倾角的测量范围取决于热电偶加速度传感器的安装方向。如果 PCB(印制电路板)上的加速度传感器是水平安装的,那么可以精确测量水平到 $\pm 45^\circ$ 的倾角。如果加速度传感器是垂直于 PCB 安装的,那么可以得到高精度的全角度的倾斜角($0^\circ \sim 360^\circ$)。垂直安装时,旋转或者倾斜的角度是一个感应轴交于 PCB 板的角度。

通过 C51 实现加速度传感器的数据采集源程序如下:

```
#include <reg52.h>
#include <stdio.h>          /*printf()函数用*/
/*A/D 转换控制特殊功能寄存器*/
/*ADC_POWER|SPEED1|SPEED0|ADC_FLAG|ADC_START|CHS2|CHS1|CHS0*/
```

```

/*ADC_POWER 必须置 1*/
/*SPEED1 | SPEED0 速度控制位*/
/*为 1, 1 时表示 210 周期转换一次; 为 1, 0 时表示 420 周期转换一次*/
/*为 0, 1 时表示 630 周期转换一次; 为 0, 0 时表示 840 周期转换一次*/
sfr ADC_CONTR = 0xC5;
/*A/D 转换结果寄存器, 高 8 位*/
sfr ADC_DATA = 0xC6; /*STC12C5410 A/D 为 8 个 10 位 A/D*/
/*A/D 转换结果寄存器, 低 2 位*/
sfr ADC_LOW2 = 0xBE;
/*A/D 特殊寄存器, 可分别设置 8 个 A/D*/
/*P1 口设置的寄存器位*/
/*当 M0=0, M1=0 时, 表示准双向口, 灌电流可达 20 mA, 拉电流为 230  $\mu$ A*/
/*当 M0=0, M1=1 时, 表示推挽输出(强上拉输出), 可达 20mA, 尽量少用*/
/*当 M0=1, M1=0 时, 表示仅为输入(高阻), 作 A/D 使用*/
/*当 M0=1, M1=1 时, 表示开漏(Open Drain), 作 A/D 使用*/
sfr P1M0 = 0x91;
sfr P1M1 = 0x92;

/*****

unsigned int Aox; /*X 轴加速度输出量, 全局变量*/
unsigned int Aoy; /*X 轴加速度输出量, 全局变量*/

/*延时函数*/
void delay(unsigned int m)
{
    while(m--);
}

*****/

名称: Initial
功能: A/D 内部寄存器初始化, P1 初始化
参数:

*****/

void Initial(void)
{
    ADC_CONTR = 0;
    ADC_CONTR |= 0x0E; /*1110 0000 设置 ADC_POWER = 1, 210 个时钟转换 1 次*/
                        /*选择 P10 作为 A/D 输入, ADC_FLAG=0*/
    delay(100); /*开 A/D 转换电源要加延时, 1 ms 以内就足够了*/
    P1M0 = 0x03; P1M1 = 0x00; /*M0=1, M1=0 时, 表示仅为输入(高阻), 作 A/D 使用*/
                        /*P1.0 和 P1.1 为 A/D 输入*/

```

```

}

/******
名称: Read_AD
功能: 得到 A/D 的值
参数: 无
Aox 和 Aoy 的 A/D 值范围为 0x0000~0x03ff
*****/

void Read_AD(void)
{
    ADC_CONTR |= 0x08;          /*开始 A/D 转换, P1.0 作为 A/D 转换通道*/
    while((ADC_CONTR & 0x10)==0); /*等待转换结束*/
    Aox = (((unsigned int)ADC_DATA)<<2)((unsigned int)(ADC_LOW2 & 0x03);
          /*得到 10 位的 A/D 值*/
    ADC_CONTR &= 0xE7;          /*把 ADC_FLAG 和 ADC_START 位清 0, 关闭 A/D*/
    ADC_CONTR |= 0x09;          /*开始 A/D 转换, P1.1 作为 A/D 转换通道*/
    delay(20);                  /*更换 A/D 转换通道要适当延时, 使输入电压稳定*/
    while((ADC_CONTR & 0x10)==0); /*等待转换结束*/
    Aoy = (((unsigned int)ADC_DATA)<<2)((unsigned int)(ADC_LOW2 & 0x03);
          /*得到 10 位的 A/D 值*/
    ADC_CONTR &= 0xE7;          /*把 ADC_FLAG 和 ADC_START 位清 0, 关闭 A/D*/
}

void main(void)                /*测试主函数*/
{
    Initial();                  /*A/D 初始化*/
    while(1)
    {
        Read_AD();
        printf("Aox=%d\n",Aox);
        printf("Aoy=%d\n",Aoy);
    }
}

```

2.6 SCA100T 高精度倾角传感器

2.6.1 硬件与功能描述

SCA100T 高精度倾角传感器是基于双轴倾角的测量器件。它具有结构简单、可靠性高、抗干扰能力强等特点, 频率响应为 8~28 Hz, 能承受 20 000 g 的机械压力。SCA100T 单轴的最大输出范围约为 $\pm 40^\circ$, 有效输出范围为 $\pm 30^\circ$ 。在采样频率为 8 Hz 及以下时, SCA100T

可获得 0.002° 的输出分辨率, 可对水平、角度和倾斜度进行精确测量, 广泛应用于地面导航、载人电梯、重型机械、轮位对准等测控系统中。

1. 主要性能特点

- (1) 双轴倾角测量, 分别为 $\pm 0.5^\circ$ ($\pm 30^\circ$)、 $\pm 1^\circ$ ($\pm 90^\circ$);
- (2) 分辨率为 0.0025° (10 Hz 带宽内, 模拟输出), 最高为 0.0008° ;
- (3) 零点温漂少于 $0.1 \text{ mg}/^\circ\text{C}$;
- (4) 模拟输出阻抗大于 $10 \text{ k}\Omega$;
- (5) 供电电源为 $+4.75 \sim +5.25 \text{ V}$, 耗电小于 6 mA ;
- (6) A/D 转换典型值时间为 $150 \mu\text{s}$;
- (7) 在时钟为 500 kHz 时, 数据传输时间的典型值为 $38 \mu\text{s}$;
- (8) 比例电压输出, 内置温度补偿;
- (9) 具有 SPI 数字接口, 或 $0.5 \sim 4.5 \text{ V}$ 模拟电压输出;
- (10) 低噪声, 工作温度范围宽;
- (11) 长期稳定性非常好, 可作加速度计用;
- (12) 外形尺寸为 $11.3 \text{ mm} \times 15.6 \text{ mm} \times 5.1 \text{ mm}$ 。

2. 内部结构与引脚说明

SCA100T 倾角传感器由敏感元件、信号处理单元、滤波器、E²PROM、校准存储器、温度传感器、A/D 转换器和 SPI 接口等组成, 如图 2-22(a)所示。其引脚采用 12 脚 DIP 封装, 见图 2-22(b)。其中:

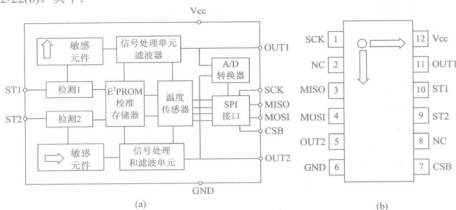


图 2-22 SCA100T 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

- (1) OUT1 和 OUT2 是 X、Y 轴角度模拟输出端;
- (2) ST1 和 ST2 是两个通道自检输入端;
- (3) SCK 是 SPI 时钟输入端;
- (4) MISO 是 SPI 数据输出端;
- (5) MOSI 是 SPI 数据输入端;
- (6) CSB 是片选端;
- (7) Vcc 和 GND 是电源与参考地。

3. SPI 串行接口

SPI 通过 4 总线同步传送数据, 其中, MOSI 是主器件(如单片机)输出、从器件(如 SCA100T)输入信号; MISO 是从器件输出、主器件输入信号; SCK 是主器件发出的串行时钟信号; CSB 是主器件控制的片选信号, 低电平有效。SPI 接口时序如图 2-23 所示。在 CSB 下降沿的作用下, 开始数据的输入/输出过程。时钟为高电平期间, 数据必须保持稳定; 时钟为低电平期间, 数据则在变化。数据传完后, CSB 必须拉回到高电平。

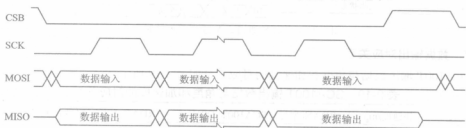


图 2-23 SPI 接口时序

SPI 接口传送数据是通过 8 位命令(或指令)进行的。SCA100T 器件 SPI 命令如表 2-18 所示。

表 2-18 SCA100T 器件 SPI 命令

命令名称	命令格式	含 义
MEAS	0000 0000(00H)	测量模式(在上电后, 默认为工作模式), 退出自检命令
RWTR	0000 1000(08H)	读/写温度寄存器命令
RDSR	0000 1010(0AH)	读状态寄存器命令
RLOAD	0000 1011(0BH)	将 E ² PROM 数据送入输出寄存器命令
STX	0000 1110(0EH)	触发 X 通道自检命令
STY	0000 1111(0FH)	触发 Y 通道自检命令
RDAX	0001 0000(10H)	读 X 通道角度(加速度)数据
RDAY	0001 0001(11H)	读 Y 通道角度(加速度)数据

读/写温度寄存器的时序如图 2-24 所示。

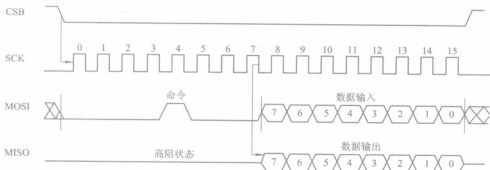


图 2-24 读/写温度寄存器的时序

读 X 通道或 Y 通道数据的时序如图 2-25 所示。

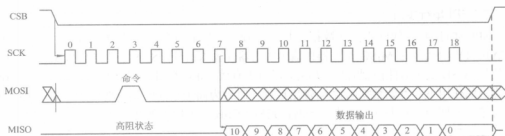


图 2-25 执行 X 通道或 Y 通道数据的时序

4. 数据输出对应关系

SCA100T 输出数据与角度和加速度的对应关系如表 2-19 所示。

表 2-19 SCA100T 输出数据与角度和加速度的对应关系

角度	加速度/mg	RDAX(型号: SCA100T-D01)	RDAX(型号: SCA100T-D02)
-5°	-87.16	DEC: 881, BIN: 011 0111 0001	DEC: 953, BIN: 011 1011 1001
-1°	-17.45	DEC: 995, BIN: 011 1110 0011	DEC: 1010, BIN: 011 1111 0010
0°	0	DEC: 1024, BIN: 100 0000 0000	DEC: 1024, BIN: 100 0000 0000
1°	17.45	DEC: 1053, BIN: 100 0001 1101	DEC: 1038, BIN: 100 0000 1110
5°	87.16	DEC: 1167, BIN: 100 1000 1111	DEC: 1095, BIN: 100 0100 0111

表中, DEC 表示十进制, BIN 表示二进制。

2.6.2 应用电路与编程

SCA100T 的应用电路如图 2-26 所示。由于该系列器件内置了一个 11 位的 A/D 转换器, 会产生周期为 50~70 μs、持续时间约 1 μs 的毛刺, 且这个毛刺被叠加到模拟信号输出端, 因此在低速转换时影响不大, 但在进行高速转换时可能引起测量错误。所以, 需要在模拟信号的输出端加上一个一阶低通滤波器, 方可有效滤除毛刺的影响。

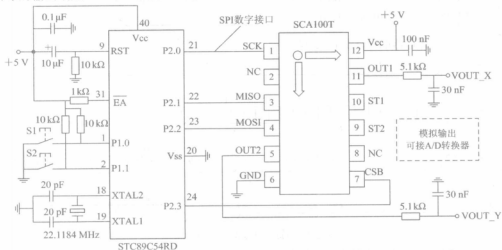


图 2-26 SCA100T 的应用电路

通过 SPI 接口用 C51 实现的采集源程序如下:

```
#include <reg52.h>
#include <stdio.h>
/*配置管脚*/
sbit SCK = P2^0;      /*SCK 时钟*/
sbit MISO= P2^1;      /*MISO 数据输出*/
sbit MOSI= P2^2;      /*MOSI 数据输入*/
sbit CSB = P2^3;      /*CSB 使能端*/
/*命令字定义*/
#define MEAS  0x00    /*测量模式(在上电后, 默认为工作模式), 退出自检命令*/
#define RWTR  0x08    /*读/写温度寄存器命令*/
#define RDSR  0x0A    /*读状态寄存器命令*/
#define RLOAD 0x0B    /*将 E2PROM 数据送入输出寄存器命令*/
#define STX    0x0E    /*触发 X 通道自检命令*/
#define STY    0x0F    /*触发 Y 通道自检命令*/
#define RDAX  0x10    /*读 X 通道角度(加速度)数据*/
#define RDAY  0x11    /*读 Y 通道角度(加速度)数据*/

void delay(void)      /*延时函数*/
{
    unsigned char m=200;
    while(m--);
}
/*SPI 写一字节函数, 参数为要写的字节*/
void SPI_writebyte(unsigned char sedata)
{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        SCK=0; delay();
        MOSI=(bit)(sedata & 0x80);
        sedata<<=1;
        SCK=1; delay();
    }
}
/*SPI 读一字节函数*/
unsigned char SPI_readbyte(void)
{
    unsigned char redata=0,i;
```



```

    for(i=0; i<8; i++)
    {
        SCK=0; delay();
        redata<<=1;
        if(MISO==1)
            redata++;          /*可用 redata=redatal0x01 替换*/
        SCK=1; delay();
    }
    return(redata);
}

/*读状态寄存器命令*/
unsigned char  rdsr(void)
{
    unsigned char ch;
    CSB=0; delay();
    SPI_writebyte(RDSR);      /*发送命令 RDSR*/
    ch=SPI_readbyte();        /*读数据*/
    delay(); CSB=1;
    return ch;
}

/*读角度(加速度)数据*/
/*参数 n 为 X 通道、Y 通道选择位*/
/*n=0 表示 X 通道角度(加速度)数据, n=1 表示 Y 通道角度(加速度)数据*/
unsigned int read_angle(bit n)
{
    unsigned int ang=0;
    unsigned char i;
    CSB=0; delay();
    if(n==0)
        SPI_writebyte(RDAX);      /*读 X 通道*/
    else SPI_writebyte(RDAY);      /*读 Y 通道*/
    for(i=0; i<11; i++)          /*读 11 位数据*/
    {
        SCK=0; delay();
        ang<<=1;
        if(MISO==1)
            ang++;
        SCK=1; delay();
    }
}

```

```
delay(); CSB=1;
return ang;
}
/*读温度寄存器命令*/
unsigned char rdtr(void)
{
    unsigned char ch;
    CSB=0; delay();
    SPI_writebyte(RWTR);          /*发送命令 RWTR*/
    ch=SPI_readbyte();            /*读数据*/
    delay(); CSB=1;
    return ch;
}
/*写温度寄存器命令*/
/*ch 为要写入的字节*/
void wrtr(unsigned char ch)
{
    CSB=0; delay();
    SPI_writebyte(RWTR);          /*发送命令 RWTR*/
    SPI_writebyte(ch);            /*读数据*/
    delay(); CSB=1;
}
void main(void)
{
    unsigned int x_angle,y_angle;
    while(1)
    {
        x_angle=read_angle(0);    /*读 X 角度(加速度)数据*/
        printf("x_angle=%d\n",x_angle);
        y_angle=read_angle(1);    /*读 Y 角度(加速度)数据*/
        printf("y_angle=%d\n",y_angle);
    }
}
```

第3章 高性能 A/D、D/A 转换器的使用与编程

3.1 ADS1110 带有内部基准的 16 位 A/D 转换器

3.1.1 硬件与功能描述

ADS1110 是一种基于 SOT23-6 小型封装、差分输入、连续自校准高精度的 16 位模拟/数字(A/D)转换器;内部带有 2.048 V 参考电源,适应 ± 2.048 V 的模拟电压输入范围;采用兼容的 I²C 串行接口协议,能在+2.7~+5.5 V 单电源下可靠工作。

ADS1110 可每秒采样 15、30、60 或 240 次。片内可编程的增益放大器(PGA)提供高达 8 倍的增益,并且允许以高分辨率对较小的信号进行测量。在单周期转换方式中,ADS1110 在一次转换之后自动掉电,在空闲期间极大地减少了电流消耗。

ADS1110 为需要高分辨率测量的应用而设计,在这种应用中,空间尺寸和电源功耗是首要考虑的问题。该器件可广泛应用于便携式仪表、工业过程控制、智能变送器、消费类产品和高分辨率测量系统中。

1. 主要性能特点

- (1) 精度为 $2.048 \text{ V} \pm 0.05\%$, 漂移为 $5 \times 10^{-6} / ^\circ\text{C}$;
- (2) 可编程的数据速率为 15~240 S/s(有的书中写为 SPS,表示次采样/秒);
- (3) 采样分辨率有四种,分别为 16 bit、15 bit、14 bit 和 12 bit;
- (4) 差分输入阻抗(典型值)为 $2.4 \text{ M}\Omega/\text{PGA}$,其中,PGA = 1, 2, 4, 8;
- (5) 可编程增益(PGA)有四种,即 1、2、4 和 8;
- (6) 转换速率有四种,即 240 S/s、60 S/s、30 S/s 和 15 S/s;
- (7) 采用 I²C 串行接口协议;
- (8) 具有连续自校准和单周期转换功能;
- (9) 内部基准为 +2.048 V,模拟电压输入范围为 $-2.048 \sim +2.048 \text{ V}$;
- (10) 电压范围为 +2.7~+5.5 V,电流小于 240 μA ;
- (11) 工作温度范围为 $-40 \sim +85^\circ\text{C}$ 。

2. 内部结构与引脚说明

ADS1110 的内部结构与引脚排列如图 3-1 所示。

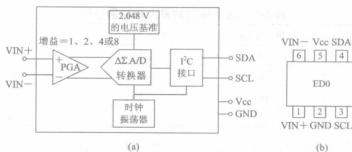


图 3-1 ADS1110 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

ADS1110 由一个带有可调增益的 $\Delta\Sigma$ /D 转换器、一个 2.048 V 的电压基准、一个时钟振荡器和一个 I²C 接口组成, 见图 3-1(a)。其引脚采用 SOT23-6 小型封装, 见图 3-1(b)。其中:

- (1) VIN+是模拟差分“+”输入端;
- (2) GND 是模拟地;
- (3) SCL 是 I²C 接口时钟输入端;
- (4) SDA 是 I²C 接口数据输入端;
- (5) Vcc 是电源输入端, 一般接+5 V;
- (6) VIN-是模拟差分“-”输入端。

由于 ADS1110 系列 A/D 转换器的器件地址在出厂前已经写入, 因此, 不同型号的器件地址标志有所不同, 表 3-1 给出了 8 种器件的型号和封装标志。

表 3-1 ADS1110 器件封装信息

器件型号	封装信息	器件地址(二进制)
ADS1110A0IDBVT/ADS1110A0IDBVR	ED0	1001 000
ADS1110A1IDBVT/ADS1110A1IDBVR	ED1	1001 001
ADS1110A2IDBVT/ADS1110A2IDBVR	ED2	1001 010
ADS1110A3IDBVT/ADS1110A3IDBVR	ED3	1001 011
ADS1110A4IDBVT/ADS1110A4IDBVR	ED4	1001 100
ADS1110A5IDBVT/ADS1110A5IDBVR	ED5	1001 101
ADS1110A6IDBVT/ADS1110A6IDBVR	ED6	1001 110
ADS1110A7IDBVT/ADS1110A7IDBVR	ED7	1001 111

3. ADS1110 的数据格式

ADS1110 是一个标准的差分输入 16 位 A/D 转换器件, 它的数据输出与模拟差分输入、内部增益(PGA)和内部 2.048 V 电压基准有关。数字输出位数和转换速率见表 3-2。

表 3-2 数字输出的最大码和最小码

转换位数/bit	转换速率/(S/s)	输出最小码	输出最大码
16	15	-32768	32767
15	30	-16384	16383
14	60	-8192	8191
12	240	-2048	2047

对最小码的最小输出值、可编程增益放大器 PGA 的增益设置以及 VIN+ 与 VIN- 的正负输入电压而言, 输出码可由以下表达式计算:

$$\text{输出码} = -1 \times \text{最小码} \times \text{PGA} \times \frac{(\text{VIN}+) - (\text{VIN}-)}{2.048}$$

在上述表达式中, 应注意使用了负的最小输出码。ADS1110 输出码的格式为二进制数 2 的补码。

ADS1110 所有的数据输出码右对齐并且经过符号扩展。在数据速率码较高时仅用一个 16 位的累加器就可进行平均值的计算。不同差分输入信号的数据输出码如表 3-3 所示。

表 3-3 不同差分输入信号的数据输出码

转换速率/(S/s)	差分输入信号				
	-2.048 V	-1 LSB	0	+1 LSB	+2.048 V
15	8000H	FFFFH	0000H	0001H	7FFFH
30	C000H	FFFFH	0000H	0001H	3FFFH
60	E000H	FFFFH	0000H	0001H	1FFFH
240	F800H	FFFFH	0000H	0001H	07FFFH

4. ADS1110 的使用

1) 工作模式

ADS1110 可以在连续转换模式或单周期转换模式下工作。在连续转换模式下, ADS1110 不断进行数据转换。一旦转换完成, 转换结果立即送入输出寄存器, 并开始下一个转换。在单周期转换模式下, ADS1110 必须等到 ST/DRDY 位在转换前被设置为“1”, 当该条件满足时, ADS1110 结束“掉电”进入转换, 转换完成后结果送入输出寄存器, 置 ST/DRDY 位为“0”, 并又进入“掉电”模式。

在 ADS1110 上电时, 内部会自动进行复位, 并将配置寄存器设置为默认方式。

2) I²C 接口

ADS1110 采用标准的 I²C 接口协议, 支持多个器件和主机共用一条总线通信。I²C 总线上的通信通常发生在两个器件之间, 其中一个作为主机, 另一个作为从机。主机和从机都能读和写, 但从机只能依主机的方向工作, ADS1110 只能作为从机使用。

ADS1110 的启动条件、结束条件和工作时序如图 1-11 所示。器件地址是 1001aaa, 其中 aaa 是厂家默认设置的, 与器件型号对应。ADS1110 的每种型号都以 EDx 为标识, 其中 x 表示地址变量。例如 ADS1110A0 标识为 ED0, 而 ADS1110A3 标识为 ED3。8 种型号见表 3-1。

ADS1110 的传送速率可以在以下三种模式下工作：第一种是标准方式，这种方式允许最高 100 kHz 的时钟频率；第二种是快速方式，这种方式允许最高 400 kHz 的时钟频率；第三种是高速方式(也称做 Hs 方式)，该方式允许最高 3.4 MHz 的时钟频率。

不需要用特殊的操作来使 ADS1110 处于标准方式或快速方式，但要采用高速方式，则必须激活该方式。为了激活高速方式，要在开始条件后发送一个特殊的地址字节 00001xxx，其中，xxx 仅适用于能采用 Hs 方式的主机。该字节称做 Hs 主机码(注意它与普通的地址字节不同，低有效位并不表明读/写状态)，ADS1110 不应答该字节。当接收到主机码时，ADS1110 将打开高速模式滤波器，并在高达 3.4 MHz 的时钟频率下通信。在下一个停止条件时，ADS1110 从 Hs 方式切换出来。

3) 寄存器的操作

ADS1110 内部有 16 位输出数据寄存器和 8 位配置寄存器。输出数据寄存器用来存储转换结果，只能通过 I²C 总线读取数据。配置寄存器用来设置器件的操作模式，如表 3-4 所示。

表 3-4 配置寄存器的设置

位	位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
位名	ST/ $\overline{\text{DRDY}}$	0	0	SC	DR1	DR0	PGA1	PGA0
缺省值	1	0	0	0	1	1	0	0

其中：

(1) ST/ $\overline{\text{DRDY}}$ 位是单次转换控制位。当 ST/ $\overline{\text{DRDY}}$ = 1 时，启动单次转换模式，单次转换结束自动使 ST/ $\overline{\text{DRDY}}$ = 0。通过查询该位的状态，可以及时读取转换的数据。在连续转换模式下，ST/ $\overline{\text{DRDY}}$ 位的值不受影响。

(2) 位 6 和位 5 是保留位。

(3) SC 位是设置连续转换模式或单次转换模式位。当 SC = 1 时，为单次转换模式；当 SC = 0 时，为连续转换模式。

(4) DR1 和 DR0 位是设置转换速率位。当 DR1、DR0 = 00 时，转换速率为 240 S/s；当 DR1、DR0 = 01 时，转换速率为 60 S/s；当 DR1、DR0 = 10 时，转换速率为 30 S/s；当 DR1、DR0 = 11 时，转换速率为 15 S/s。

(5) PGA1、PGA0 位是设置放大倍数位。当 PGA1、PGA0 = 00 时，增益为 1；当 PGA1、PGA0 = 01 时，增益为 2；当 PGA1、PGA0 = 10 时，增益为 4；当 PGA1、PGA0 = 11 时，增益为 8。

4) ADS1110 的读操作

对 ADS1110 可以通过 I²C 总线协议，一次读出输出寄存器(16 位)和配置寄存器(8 位)三字节的内容。如果不要求一定要读出配置寄存器字节的值，则在读操作中允许读出的字节个数少于三个。

从 ADS1110 中读取多于三个字节的值是无效的，从第四个字节开始的所有字节将为 FFH。

可以忽略 ST/ $\overline{\text{DRDY}}$ 位，并且可在任何时候从 ADS1110 的输出寄存器中读取数据。不管一次新的转换是否完成，如果在一个转换周期内对输出寄存器的读操作不止一次，则输出寄存器每次将返回相同的数据。只有当输出寄存器被更新时，才会返回新的数据。

ADS1110 典型的读时序如图 3-2 所示。

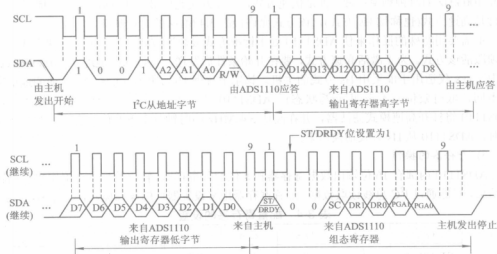


图 3-2 ADS1110 典型的读时序

5) ADS1110 的写操作

为了对配置寄存器进行写操作，要对 ADS1110 寻址，并写入一个字节。这个字节只能写入配置寄存器中，但不能将数据写入数据寄存器。对 ADS1110，若写入多个字节，则无效，器件将忽略第一个字节以后的任何输入数据，并且它只对第一个字节作出应答。ADS1110 典型的写时序如图 3-3 所示。

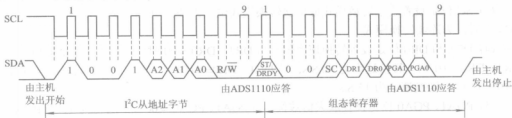


图 3-3 ADS1110 典型的写时序

3.1.2 应用电路与编程

1. 基本连接方法

对于多数应用而言，ADS1110 的连接方法非常简单。ADS1110 的基本连接图如图 3-4(a) 所示。ADS1110 的完全差分电压输入非常适应于连接到阻抗较低的差分源，如电桥传感器。尽管 ADS1110 可读取两极差分信号，但它不能接收输入端的负电压。ADS1110 可与标准方式、快速方式和高速方式的 I²C 控制器直接接口。任何微控制器的 I²C 外围设备，包括只能用作主机的设备均可与 ADS1110 一起工作。

上拉电阻对 SDA 和 SCL 线都是必要的，因为 I²C 总线驱动器是漏极开路驱动器，这些

电阻的大小取决于总线的工作速度和总线容性负载。阻值较高的电阻其功耗较低,但会延长总线的传输时间,限制总线的速度。总线在阻值较低时可高速运转,但功耗较高。但若使用长总线,则因电容较大,故需要较小的上拉电阻来补偿。一般电阻不应太小,如果电阻太小,则总线驱动器可能不能将总线拉低。

2. 连接多个器件

连接多个 ADS1110 到同一条总线是常用的方法。ADS1110 有 8 种不同类型,每种类型都有一个不同的 I²C 地址,两个 ADS1110 连接到同一条总线的接线如图 3-4(b)所示。通过这种方法,最多可以连接 8 个不同地址的 ADS1110。注意,每条总线仅需一组上拉电阻,但上拉电阻的阻值要适当小一点,以补偿由于多个器件带来的附加的总线电容和增加总线长度的影响。

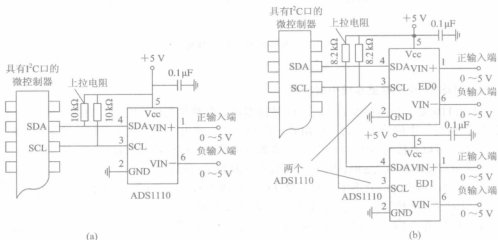


图 3-4 ADS1110 的两种典型接法

(a) 基本连接电路; (b) 多个器件连接电路

注意: 图 3-4 中, 差分信号的范围为 $-2.048 \sim +2.048$ V。

在这种多器件连接方法中, 也可以在总线上混接其他 I²C 接口的器件, 例如接入 ADS1110、A/D、D/A 和集成温度传感器等器件。

3. 用通用 I/O 口代替 I²C 接口

如果微控制器没有专用的 I²C 控制器, 则可通过软件将通用 I/O 口设置成 I²C 接口。编程时, 通过软件可模拟 I²C 总线协议或产生位脉冲, 将单个 ADS1110 连接到通用 I/O 口上, 如图 3-5(a)所示。这种接法只对 SDA 线上拉, 不需要上拉 SCL 线。

4. 单端输入

尽管 ADS1110 有一个完全差分输入端, 但它也可容易地测量出单端信号。图 3-5(b)为简单的单端连接示意图。ADS1110 通过将其任一输入引脚(通常是 VIN-)接地, 并加输入信号到 VIN+来进行单端测量。单端信号的范围是 $0 \sim 2.048$ V。不能加负电压到该引脚上, 因为 ADS1110 的输入端只接收正电压。

ADS1110 的输入范围是相对于基准电压(即 2.048 V)的两极差分电压。图 3-5(b)所示的单端电路仅涵盖了 ADS1110 输入范围的一半, 由于它没有产生差分负输入, 因此遗失一位分辨率。

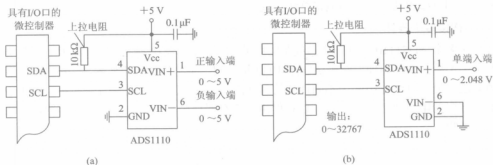


图 3-5 两种 ADS1110 连接电路

(a) 通用 I/O 口连接; (b) 单端输入电路

5. ADS1110 在称重系统中的应用

ADS1110 是一个高精度的 $\Delta\Sigma$ A/D 转换器, 适合测量变化速度不大的物理量, 例如测量温度、压力、水位值等。对于压阻式、应变片式、热电阻式、电位器式等输出阻抗较低的传感器, 可以直接与 ADS1110 连接。多数电桥结构的传感器满量程输出电压只有几十毫伏, 当满量程输出电压为 ± 30 mV 时, 用 ADS1110 测量可以达到 12 位分辨率(不包括 1 位符号位), 若要达到更高的分辨率, 则需要外接测量放大器。图 3-6 是一个称重测量系统, 传感器采用应变式电桥结构。电桥的输出经过差分放大电路, 并与 ADS1110 的全差分输入相连。A/D 转换器的 I²C 接口受 STC89C52 单片机控制。该系统具有结构简单、灵敏度高、抗干扰能力强等特点, 可以测量 0~2 t 的重量信号。

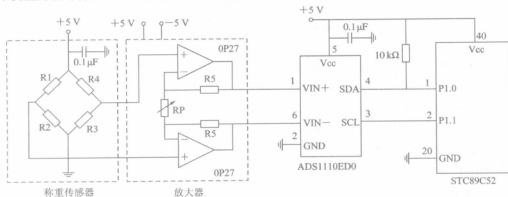


图 3-6 称重测量系统

用 C51 实现的相关采集程序如下:

```
#include "reg52.h"
sbit SCL=P1^1;          /*时钟端口*/
```

```

s bit SDA=P1^0;          /*数据端口*/
void delay(unsigned char x) /*延时函数*/
{ unsigned char i;
  for (i=0; i<x; i++);
}
void start_I2C(void)       /*启动函数*/
{
  SDA=1; SCL=1;
  delay(5); SDA=0;
  delay(5); SCL=0;
}
void stop_I2C(void)        /*停止函数*/
{
  SDA=0; SCL=0;
  delay(5); SCL=1;
  delay(5); SDA=1;
}

```

/*-----*/

功能：向 ADS1110 写入一个字节(包括总线初始化、起始与终止条件)

参数：word 为器件地址，byte 为写入配置寄存器的数据

-----*/

```

void Write_ADS1110_i2c(unsigned char word, unsigned char byte)
{
  unsigned char dat,j,ack; /*分别用于传递形参，保存读回的 ACK*/
  start_I2C();             /*启动*/
  dat=word;                /*发送 EDx 写地址*/
  for(j=0; j<8; j++)
  {
    if(dat&0x80) SDA=1;
    else SDA=0;
    delay(2); SCL=1; /*时钟线=1*/
    delay(2); SCL=0; /*时钟线=0*/
    delay(2); dat<<=1; /*左移1位*/
  }
  SDA=1; /*立即释放数据线*/
  delay(2); SCL=1; /*第9个时钟*/
  if(SDA==0) ack=0; /*接收 ACK*/
  else ack=1;
  SCL=0;
}

```

```

dat=byte;          /*发送配置寄存器的内容*/
for(j=0; j<8; j++)
{
    if(dat&0x80) SDA=1;
    else SDA=0;
    delay(2); SCL=1; /*时钟线=1*/
    delay(2); SCL=0; /*时钟线=0*/
    delay(2); dat<<=1; /*左移 1 位*/
}
SDA=1; delay(2); SCL=1;
if(SDA==0) ack=0;
else ack=1;
SCL=0;
stop_I2C();        /*产生停止条件*/
}

/*-----
功能: 从 ADS1110 读出一个整型数据(两个字节)作为返回值
参数: word 为器件读地址, *p 为带回的配置寄存器内容
-----*/

unsigned int Read_ADS1110_i2c(unsigned char word, unsigned char *p)
{
    unsigned char dat,j,ack; /*分别用于传递形参、8 位字长计数、保存读回的 ACK*/
    unsigned char data_h=0,data_l=0,data_p=0; /*得到的数据*/
    unsigned int y;
    start_I2C();          /*启动*/
    dat=word;              /*发送写地址*/
    for(j=0; j<8; j++)
    {
        if(dat&0x80) SDA=1;
        else SDA=0;
        delay(2); SCL=1; /*时钟线=1*/
        delay(2); SCL=0; /*时钟线=0*/
        delay(2); dat<<=1; /*左移 1 位*/
    }
    SDA=1; delay(2); SCL=1;
    if(SDA==0) ack=0;
    else ack=1;
    SCL=0;
    for(j=0; j<8; j++)    /*读高 8 位数据*/

```

```

{   data_h<=1; delay(2);           /*延时, 让从器件准备数据*/
    SCL=1; delay(2);               /*在 SCL 为高电平的情况下读 SDA*/
    if(SDA==0)   data_h &= 0xfe;   /*data_h 末位清 0*/
    else   data_h |= 0x01;         /*data_h 末位置 1*/
    SCL=0;                               /*时钟线置低*/
}

/*发送 ack, 通知从机收到数据*/
SCL=0; delay(2);
SCL=1; delay(2);
SDA=0; delay(2);           /*回答*/
SCL=0;

for(j=0; j<8; j++)          /*读低 8 位数据
{   data_l<=1; delay(2);     /*延时, 让从器件准备数据*/
    SCL=1; delay(2);         /*在 SCL 为高电平的情况下读 SDA*/
    if(SDA==0)   data_l &= 0xfe; /*data_l 末位清 0*/
    else   data_l |= 0x01;     /*data_l 末位置 1*/
    SCL=0;                               /*时钟线置低*/
}

/*发送 ack, 通知从机收到数据*/
SCL=0; delay(2);
SCL=1; delay(2);
SDA=0; delay(2);           /*回答*/
SCL=0;

for(j=0; j<8; j++)          /*读配置寄存器内容*/
{   data_p<=1; delay(2);     /*延时, 让从器件准备数据*/
    SCL=1; delay(2);         /*在 SCL 为高电平的情况下读 SDA*/
    if(SDA==0)   data_p &= 0xfe; /*data_p 末位清 0*/
    else   data_p |= 0x01;     /*data_p 末位置 1*/
    SCL=0;                               /*时钟线置低*/
}

/*发送 ack, 通知从机收到数据*/
SCL=0; delay(2);
SCL=1; delay(2);
SDA=0; delay(2);           /*回答*/
SCL=0;

*p=data_p;                   /*带回配置寄存器内容*/
y=(unsigned int)data_h*256+(unsigned int)data_l;
stop_J2C0;                   /*产生停止条件*/
return(y);                   /*返回 A/D 的值*/

```

```

    }
void main(void)                                /*主函数，调试用*/
{ unsigned char ad_state;                      /*A/D 转换模式*/
  unsigned char ad_word;                       /*A/D 转换器的器件地址*/
  unsigned int result;                         /*结果数据*/
  unsigned char *p1;                           /*配置数据*/
  ad_state=0x0c;                               /*连续转换，16 bit，增益=1*/
  ad_word=0x90;                                /*ED0 器件*/
  Write_ADS1110_i2c(ad_word, ad_state);        /*设置 A/D 的状态*/
  result=Read_ADS1110_i2c(ad_word|0x01, p1);   /*读数据*/
  while(1);                                   /*等待*/
}

```

3.2 MCP3221 低成本低功耗 12 位 A/D 转换器

3.2.1 硬件与功能描述

MCP3221 为单端输入、逐次逼近、小封装、12 位 A/D 转换器。基于先进的 CMOS 技术，MCP3221 可分别提供 250 μA 和 1 μA 的最大转换电流和待机电流。由于其电流消耗很低，且采用小型 SOT-23 封装形式，因此该器件非常适用于电池供电。

MCP3221 采用 2 线制 I²C 总线接口，具有标准 100 kHz 和快速 400 kHz 两种通信模式，内部转换时钟和外部转换时钟相互独立，允许在两条总线上最多挂接 8 个器件。

该器件可广泛应用于数据记录、多区域监视、手持便携式仪表和远程遥控的数据采集系统。

1. 主要性能特点

- (1) 单端模拟输入，内部具有采样/保持电路；
- (2) 模拟输入采集时间的典型值为 1.12 μs ；
- (3) A/D 转换时间的典型值为 8.96 μs ；
- (4) 在 I²C 快速模式下的转换速率为 22.3 kS/s；
- (5) A/D 分辨率为 12 位；
- (6) 采用 I²C 接口，内部具有转换时钟电路；
- (7) 待机电流的典型值为 5 nA，最大为 1 μA ；
- (8) 转换电流最大为 250 μA ；
- (9) 单电源供电，范围为 2.7~5.5 V；
- (10) 工业级温度范围为 -40~+85℃，可扩展到 +125℃；
- (11) 小型 SOT-23 封装。

2. 内部结构与引脚说明

MCP3221 的内部结构与引脚排列如图 3-7 所示。它由输入电路、采样/保持电路、时钟

电路、12 位 D/A 转换电路、逐次逼近电路、比较电路、控制逻辑电路和 I²C 接口电路等组成, 见图 3-7(a)。其引脚采用 SOT-23 小型封装, 见图 3-7(b)。其中:

- (1) V_{CC} 是电源端, 一般接+5 V;
- (2) GND 是模拟地;
- (3) VIN 是模拟信号输入端;
- (4) SDA 是 I²C 接口数据输入/输出端(漏极开路);
- (5) SCL 是 I²C 接口的时钟端(漏极开路)。

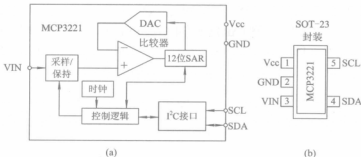


图 3-7 MCP3221 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

3. 工作原理

MCP3221 采用了经典的 SAR 架构。发生转换时, 该器件使用内部采样保持电容来保持模拟输入电平。在采集结束时, 转换器的输入开关打开, 采样保持电容上的电荷产生一个 12 位的串行数字输出编码。采集和转换时间可使用内部时钟自行设定。每次转换完成后, 就会将结果存储到一个 12 位寄存器中, 并可随时读取。通过一个 2 线 I²C 接口实现与处理器之间的通信。在连续转换模式下, 若 SCL 时钟速率为 400 kHz, 则 MCP3221 可实现的最大采样速率为 22.3 kS/s。

1) 数字输出编码

由 MCP3221 产生的数字输出编码是输入信号(VIN)和电源电压(V_{CC})的函数。随着 V_{CC} 电平的降低, 数字输出的二进制数(LSB)的大小也会相应地减少。理论上 LSB 的大小为

$$\text{LSB} = \frac{V_{\text{CC}}}{4096}$$

2) 转换时间

转换时间 T_{CONV} 是指在模拟输入与保持电容断开后, 得到数字结果所需的时间。对于 MCP3221, 规定的转换时间通常为 8.96 μs 。该时间取决于内部振荡器, 与 SCL 无关。

3) 采集时间

采集时间 T_{ACQ} 是指采样电容阵列采集电荷所需的时间。采集时间通常为 1.12 μs 。该时间取决于内部振荡器, 与 SCL 无关。

4) 采样速率

采样速率是指从第一次转换采集开始到第二次转换采集开始所需要的最大时间的倒

数。可以通过单次或连续转换测得采样速率。单次转换包含一个启动位、一个地址字节、两个数据字节和一个停止位。采样速率是在两个启动位之间测得的。在两次转换之间或全部 18 个时钟周期(两个数据字节和两个应答位)内可测得最大采样速率。

4. 串行通信

1) I²C 总线特性

I²C 总线协议规定:

- 只有当总线不忙时才会启动数据传输。
- 传输数据时,若时钟线为高电平,则数据线必须保持稳定。在时钟线为高电平时,改变数据线的电平将被视为启动(START)或结束(STOP)条件。典型时序如图 3-8 所示。

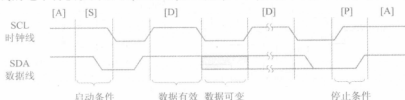


图 3-8 典型时序图

- (1) 总线不忙[A]。数据和时钟线均保持高电平。
- (2) 启动数据传输[S]。当时钟(SCL)为高电平时, SDA 线上从高到低的跳变定义为启动(START)条件。所有命令必须在 START 条件出现后有效。
- (3) 停止数据传输[P]。当时钟(SCL)为高电平时, SDA 线上从低到高的跳变定义为停止(STOP)条件。所有操作都以 STOP 条件结束。
- (4) 有效数据[D]。START 条件后,如果在时钟信号为高电平期间,数据线保持稳定,则此时数据线上的状态代表的是有效数据。在时钟信号为低电平期间,必须更改数据线上的数据。每个数据位均对应一个时钟脉冲。每次数据传输都以 START 条件开始,以 STOP 条件结束。在 START 和 STOP 条件之间传输的数据字节数由主器件决定,长度没有限制。
- (5) 应答。被寻址的接收器件每接收到一个字节就必须产生一个应答位。主器件必须产生一个与应答位相关的额外时钟脉冲。

应答器件要在应答时钟脉冲期间将 SDA 线拉为低电平,即在与应答相关的时钟脉冲为高电平期间保持 SDA 线为低电平,必须将建立和保持时间考虑在内。读操作期间,主器件必须通过在接收到来自从器件的最后一个字节时,不产生应答(NAK)来向从器件发出数据结束信号。在这种情形下,从器件(MCP3221)将释放总线,以允许主器件产生 STOP 条件。

MCP3221 支持一个双向 2 线总线和数据传输协议。将数据发送到总线的器件为发送器,而接收数据的器件是接收器。当 MCP3221 作为从器件工作时,必须由主器件控制总线,由其产生串行时钟(SCL)、控制总线的访问权,并产生 START 和 STOP 条件。主器件和从器件都可以作为发送器或接收器,但由主器件决定工作模式。

2) 器件寻址

地址字节是 START 条件后从主器件接收到的第一个字节。控制字节的第一部分由 4 位器件代码组成,对于 MCP3221,该代码被设置为 1001。器件代码后跟有三个地址位: A2、A1 和 A0,默认地址位是 101(可从 Microchip 公司获取其他地址位选项,器件地址与型号见

表 3-5)。其他地址位允许一条总线上最多可连接 8 个 MCP3221 器件，并可用于确定被访问的器件。从地址的第 8 位决定主器件是要读取转换数据还是对 MCP3221 执行写操作。当该位置“1”时，选择读操作；当该位置“0”时，选择写操作。MCP3221 中没有可写入的寄存器，所以必须将该位置“1”以启动转换。MCP3221 是与 2 线 I²C 串行接口协议兼容的从器件。

表 3-5 MCP3221 型号与器件地址

器件型号	地址	SOT-23	器件型号	地址	SOT-23
MCP3221A0T-I/OT	000	EE	MCP3221A0T-E/OT	000	GE
MCP3221A1T-I/OT	001	EH	MCP3221A1T-E/OT	001	GH
MCP3221A2T-I/OT	010	EB	MCP3221A2T-E/OT	010	GB
MCP3221A3T-I/OT	011	EC	MCP3221A3T-E/OT	011	GC
MCP3221A4T-I/OT	100	ED	MCP3221A4T-E/OT	100	GD
MCP3221A5T-I/OT	101	S1*	MCP3221A5T-E/OT	101	GA
MCP3221A6T-I/OT	110	EF	MCP3221A6T-E/OT	110	GF
MCP3221A7T-I/OT	111	EG	MCP3221A7T-E/OT	111	GG

3) 启动采样和保持

从地址字节“R/W = 1”位的下降沿启动输入信号的采集和转换。从这一刻起，由内部时钟启动采样、保持和转换周期，所有这些都是在 ADC 内部实现的。在发送了一个为逻辑高电平的“R/W = 1”位后，在时钟的第一个下降沿开始采样输入信号。此外，ADC 将在 SCL 的上升沿发送一个应答位(ACK = 0)。主器件必须在 ADC 发送应答的时钟脉冲期间释放数据总线，以允许 MCP3221 将该线拉低。

对于连续采样的情况，从转换结果(两个字节)的 LSB 的下降沿继续采样，如图 3-10 所示。

4) 读取转换数据

MCP3221 应答地址字节之后，发送器将发送 4 个 0，后跟转换结果的高 4 位。然后主器件使用 ACK = “0”来应答该字节。MCP3221 将在随后出现的 8 个时钟脉冲期间发送低 8 位转换结果。然后主器件发送 ACK = “1”告知 MCP3221 其不再请求更多的数据。最后主器件发送一个停止位终止发送。读取转换数据时序如图 3-9 所示。

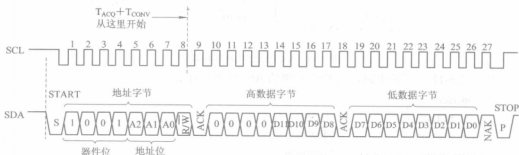


图 3-9 MCP3221 读时序

5) 连续读取数据

对于连续采样, 从转换结果 LSB(最低位)的下降沿继续采样。有关时序信息如图 3-10 所示。图中, $T_{ACQ}+T_{CONV}$ 表示采集和转换时间。

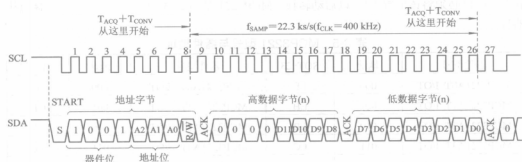


图 3-10 MCP3221 连续读时序

3.2.2 应用电路与编程

MCP3221 12 位 A/D 转换器与 STC89C58RC 单片机的连接电路如图 3-11 所示。因 I^2C 总线是一个漏极开路的总线, 故 SDA 和 SCL 线必须接上拉电阻。为了减少总线上的分布电容的影响, 通常时钟在 100 kHz 以内, 上拉电阻取 10 k Ω , 如果时钟为 400 kHz, 则上拉电阻应取 2 k Ω 。

图 3-11 中, 在电源线上一般要用 0.1 μF 的旁路电容。如果电源波纹较大, 则还需在该旁路电容上并联一个 10 μF 的电容以削弱高频噪声。

考虑到 MCP3221 是一个单极性(0 V~+Vcc)的 A/D 转换器, 要将其变成双极性输入, 图中设计了同相缓冲器, 并通过两个 10 k Ω 电阻使 0~+5 V 输入变成-5~+5 V 输入。

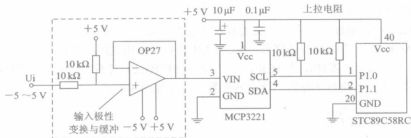


图 3-11 MCP3221 的应用电路

对于图 3-11 所示的电路, 由 C51 实现的 A/D 转换程序如下:

```
#include "reg51.h"
s bit SCL=P1^0;          /*时钟端口*/
s bit SDA=P1^1;          /*数据端口*/
void delay(unsigned char x) /*延时函数*/
{
    ;
}
```

```

    unsigned char i;
    for (i=0; i<x; i++){
    }
void start_I2C(void)          /*启动函数*/
{
    SDA=1; SCL=1;
    delay(5); SDA=0;
    delay(5); SCL=0;
}
void stop_I2C(void)          /*停止函数*/
{
    SDA=0; SCL=0;
    delay(5); SCL=1;
    delay(5); SDA=1;
}
/*-----*/
功能：从MCP3221读出一个整型数据(两个字节)作为返回值
参数：word为器件读地址
-----*/
unsigned int Read_MCP3221_i2c(unsigned char word)
{
    unsigned char dat,i,ack; /*分别用于传递形参、8位字长计数、保存读回的ACK*/
    unsigned char data_h=0,data_l=0;          /*得到的数据*/
    unsigned int y;
    start_I2C();                               /*启动*/
    dat=word;                                  /*发送写地址*/
    for(i=0; i<8; i++)
    {
        if(dat&0x80) SDA=1;                    /*高位在前*/
        else SDA=0;
        delay(2); SCL=1;
        delay(2); SCL=0;
        delay(2); dat<<=1;
    }
    SDA=1; delay(2); SCL=1;
    if(SDA==0) ack=0;
    else ack=1;
    SCL=0;
    for(i=0; i<8; i++)                          /*读高8位数据*/
    {
        data_h<=1; delay(2);                    /*左移位，延时，让从器件准备数据*/
        SCL=1; delay(2);                        /*在SCL为高电平的情况下读SDA*/
    }
}

```

```

        if(SDA==0) data_h &= 0xfe;          /*data_h末位清0*/
        else data_h |= 0x01;                /*data_h末位置1*/
        SCL=0;                             /*时钟线置低*/
    }
    /*发送ack, 通知从机收到数据*/
    SCL=0; delay(2);
    SCL=1; delay(2);
    SDA=0; delay(2);
    SCL=0;
    for(i=0; i<8; i++)                      /*读低8位数据*/
    { data_l<=1; delay(2);                  /*左移位, 延时, 让从器件准备数据*/
      SCL=1; delay(2);                     /*在SCL为高电平的情况下读SDA*/
      if(SDA==0) data_l &= 0xfe;           /*data_l末位清0*/
      else data_l |= 0x01;                 /*data_l末位置1*/
      SCL=0;                               /*时钟线置低*/
    }
    /*发送ack, 通知从机收到数据*/
    SCL=0; delay(2);
    SCL=1; delay(2);
    SDA=0; delay(2);
    SCL=0;
    y=(unsigned int)data_h*256+(unsigned int)data_l;
    /*将两个8位数据拼成一个整型变量*/
    stop_I2C();                             /*产生停止条件*/
    return(y);                              /*返回A/D的值*/
}

void main(void)                            /*调试主函数*/
{
    unsigned char ad_word;                  /*A/D转换器的器件地址*/
    unsigned int result;                   /*结果数据*/
    result=Read_MCP3221_i2c(0x9b);        /*读取12 bit数据*/
    while(1);
}

```

3.3 AD7705/06 用于低频测量的 16 位 A/D 转换器

3.3.1 硬件与功能描述

AD7705/06 是应用于低频测量的 2/3 通道的模拟前端器件。该器件可以直接连接来自传感器的低电平输入信号, 然后产生串行的数字输出。该器件利用 $\Sigma\Delta$ 转换技术可实现 16 位

无丢失代码性能。

AD7705 是双通道全差分模拟输入, 而 AD7706 是 3 通道差分模拟输入, 两者都有一个差分基准输入单元。当电源电压为 5 V、基准电压为 2.5 V 时, 这两种器件都可将输入信号范围为 0~20 mV 和 0~2.5 V 的信号进行处理, 还可处理 $\pm(20 \text{ mV} \sim 2.5 \text{ V})$ 的双极性输入信号。AD7705 以 $\text{AIN}(-)$ 输入端为参考点, 而 AD7706 以模拟地为参考点。当电源电压为 3 V, 基准电压为 1.225 V 时, 可处理 0~10 mV 和 0~1.225 V 的单极性输入信号, 它的双极性输入信号范围是 $\pm 10 \text{ mV} \sim \pm 1.225 \text{ V}$ 。因此, AD7705/06 可以实现 2/3 通道系统所有信号的调理和转换。

AD7705/06 的串行接口可配置为 3 线接口。增益值、信号极性和更新速率的选择可由软件设置。该器件还包括自校准和系统校准选项, 以消除器件本身所造成的偏移误差。

AD7705/06 采用 16 脚双列直插(DIP)、16 脚宽体(0.3 英寸)SOIC 封装和 16 脚 TSSOP 封装。

1. 主要性能特点

- (1) 非线性: $<0.003\%$;
- (2) 增益在 1~128 范围内可调;
- (3) 待电流小于 $8 \mu\text{A}$;
- (4) 差分或单端模拟输入, 单极性为 0~20 mV 到 0~2.5 V, 双极性为 $\pm(20 \text{ mV} \sim 2.5 \text{ V})$;
- (5) 差分基准输入;
- (6) 内部提供数字滤波;
- (7) SPI、QSPI 或 DSP 串行接口;
- (8) 只需 2.7~3.3 V 或 4.75~5.25 V 的单电源;
- (9) 具有极低的功耗, 在掉电模式时, 功耗的典型值为 $20 \mu\text{W}$ 。

2. 内部结构与引脚说明

AD7705/06 的内部结构与引脚排列如图 3-12 所示。其引脚功能见表 3-6。

表 3-6 AD7705/06 的引脚功能

引脚	名称	功能描述
1	SCLK	串行时钟输入信号
2, 3	CLKi, CLKo	时钟信号输入/输出(一般接晶振), 500 kHz~5 MHz
4	$\overline{\text{CS}}$	片选, 低电平有效
5	RESET	复位输入, 低电平有效
6	IN2+或 IN1	差分模拟输入 2 的正输入端或单端模拟 1 输入
7	IN1+或 IN2	差分模拟输入 1 的正输入端或单端模拟 2 输入
8	IN1-或 COM	差分模拟输入 1 的负输入端或单端模拟参考“地”输入
9	REF+	基准“+”输入
10	REF-	基准“-”输入
11	IN2-或 IN3	差分模拟输入 2 的负输入端或单端模拟 3 输入
12	$\overline{\text{DRDY}}$	逻辑输出, 可以获取新的输出字, 可用来指示何时完成校准
13	DOUT	串行数据输出端
14	DIN	串行数据输入端
15, 16	Vcc, GND	电源(+2.7~+5.25 V)与参考地

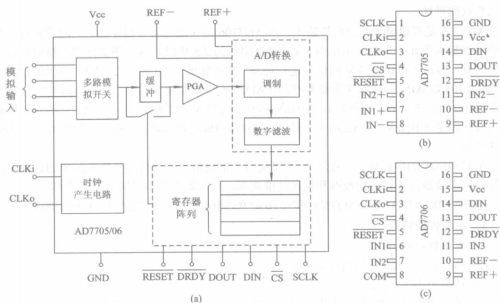


图 3-12 内部结构与引脚排列

(a) 内部结构; (b) AD7705 的引脚排列; (c) AD7706 的引脚排列

3. 内部电路说明

1) 片内寄存器

AD7705/06 内部包括 8 个寄存器, 这些寄存器通过器件的串行口访问。第 1 个是通信寄存器, 它管理通道选择, 决定下一个操作是读操作还是写操作, 以及下一次读或写哪一个寄存器。所有与器件的通信必须从写入通信寄存器开始。此外, 通信寄存器还控制等待模式和通道选择, $\overline{\text{DRDY}}$ 状态也可以从通信寄存器上读出。第 2 个寄存器是设置寄存器, 决定校准模式、增益设置、单/双极性输入以及缓冲模式。第 3 个寄存器是时钟寄存器, 包括滤波器选择位和时钟控制位。第 4 个是数据寄存器, 器件输出的数据从这个寄存器读出。最后一个是校准寄存器, 它存储通道校准数据。

(1) 通信寄存器(RS2、RS1、RS0=000, 上电/复位默认值为 00H)。通信寄存器是一个 8 位寄存器, 既可以读出数据也可以把数据写进去。所有与器件的通信必须从写该寄存器开始。写入的数据决定下一次读操作或写操作在哪个寄存器上发生。一旦在选定的寄存器上完成了下一次读操作或写操作, 接口将返回到写操作的状态, 这也是接口的默认状态。在上电或复位后, AD7705/06 就处于这种默认状态等待对通信寄存器进行一次写操作。在接口序列丢失的情况下, 如果在 DIN 高电平的写操作持续了足够长的时间(至少 32 个串行时钟周期), 则 AD7705 将会回到默认状态。通信寄存器的格式如表 3-7 所示。

表 3-7 通信寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
0/ $\overline{\text{DRDY}}$ (0)	RS2(0)	RS1(0)	RS0(0)	R/W(0)	STBY(0)	CH1(0)	CH0(0)

表 3-7 中, 括号内的值为上电复位的缺省值。

0/DRDY: 对于写操作, 必须将“0”写到该位, 以便能实现其他各位的写入。如果将“1”写到该位, 则后续各位将不能写入该寄存器, 它会停留在该位直到有一个“0”被写入该位。对于读操作, 该位提供器件的 **DRDY** 标志。该位的状态与 **DRDY** 输出引脚的状态相同。

RS2~RS0: 寄存器选择位。这 3 个位用来选择下次“读/写”操作在 8 个片内寄存器中的哪一个上进行, 见表 3-8。当选定的寄存器完成了读/写操作后, 器件返回到等待通信寄存器下一次写操作的状态。这 3 个位不会保持为继续访问原寄存器的状态。

表 3-8 寄存器的选择

RS2	RS1	RS0	寄存器名	寄存器位数
0	0	0	通信寄存器	8 位
0	0	1	设置寄存器	8 位
0	1	0	时钟寄存器	8 位
0	1	1	数据寄存器	16 位
1	0	0	测试寄存器	8 位
1	0	1	无操作	
1	1	0	偏移寄存器	24 位
1	1	1	增益寄存器	24 位

R/W: 读/写选择。这个位用于选择下次操作对选定的寄存器是读还是写。“0”表示下次操作是写, “1”表示下次操作是读。

STBY: 等待模式。此位上写“1”, 则处于等待或掉电模式。在这种模式下, 器件消耗的电源电流仅为 10 μ A。在等待模式时, 器件将保持它的校准系数和控制字信息。若此位上写“0”, 则器件处于正常工作模式。

CH1、CH0: 通道选择。这 2 个位选择一个通道, 以供数据转换或访问校准系数。器件内的 3 对校准寄存器用来存储校准系数。表 3-9 列出了具有独立的校准系数的通道组合。当 CH1 为逻辑 1 而 CH0 为逻辑 0 时, 由表可见 AD7705 是 IN1-输入脚在内部自己短路, 而 AD7706 是 COM 脚在内部自己短路。这可以作为评估噪声性能的一种测试方法(无外部噪声源)。在这种模式下, IN1-/COM 输入端必须与一个器件允许的共模电压范围内的外部电压相连接。

表 3-9 AD7705/06 的通道选择

AD7705					AD7706				
CH1	CH0	IN+	IN-	校准寄存器	CH1	CH0	IN	基准	校准寄存器
0	0	IN1+	IN1-	寄存器对 0	0	0	IN1	COM	寄存器对 0
0	1	IN2+	IN2-	寄存器对 1	0	1	IN2	COM	寄存器对 1
1	0	IN1-	IN1-	寄存器对 0	1	0	COM	COM	寄存器对 0
1	1	IN2-	IN2-	寄存器对 2	1	1	IN3	COM	寄存器对 2

(2) 设置寄存器(RS2、RS1、RS0=001, 上电/复位默认值为 01H)。设置寄存器是一个 8 位寄存器, 它既可以读数据又可将数据写入。设置寄存器的格式见表 3-10。

表 3-10 设置寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
MD1(0)	MD0(0)	G2(0)	G1(0)	G0(0)	B/U(0)	BUF(0)	FSYNC(1)

MD1、MD0 的设置见表 3-11。

表 3-11 MD1、MD0 的设置

MD1	MD0	工 作 模 式
0	0	正常模式。在这种模式下, 转换器进行正常的模/数转换
0	1	自校准。在通信寄存器的 CH2 和 CH1 选中的通道上激活自校准
1	0	零标度系统校准。在通信寄存器的 CH2 和 CH1 选中的通道上激活零标度校准
1	1	满标度系统校准。在选定的输入通道和在选定的增益下激活满标度系统校准

G2~G0: 增益选择位, 设置内部 PGA 的增益。其中, 二进制的 000~111 表示 1~128 个增益。

B/U: 单极性/双极性工作选择。“0”表示选择双极性操作, “1”表示选择单极性工作。

BUF: 缓冲器控制。“0”表示片内缓冲器短路, 缓冲器短路后, 电源电流降低。此位处于高电平时, 缓冲器与模拟输入串联, 输入端允许处理高阻抗源。

FSYNC: 滤波器同步。该位为“1”, 数字滤波器的节点、滤波器控制逻辑和校准控制逻辑处于复位状态, 同时, 模拟调制器也被控制在复位状态下。当该位为“0”时, 调制器和滤波器开始处理数据。FSYNC 的值不影响数字接口, 也不使 $\overline{\text{DRDY}}$ 输出复位(如果是低电平)。

(3) 时钟寄存器(RS2、RS1、RS0=010, 上电/复位默认值为 05H)。时钟寄存器是一个可以读/写数据的 8 位寄存器。时钟寄存器的格式见表 3-12。

表 3-12 时钟寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
ZERO(0)	ZERO(0)	ZERO(0)	CLKDIS(0)	CLKDIV(0)	CLK(1)	FS1(0)	FS0(1)

ZERO: 必须在这些位上写零, 以确保 AD7705/06 的正确操作。否则, 会导致器件出错。

CLKDIS: 主时钟禁止位。逻辑“1”表示阻止主时钟在 CLK_O 引脚上输出。禁止时, CLK_O 引脚处于低电平。

CLKDIV: 时钟分频器位。CLKDIV 置为逻辑“1”时, CLK_I 引脚处的时钟频率在被 AD7705/06 使用前进行 2 分频。例如, 将 CLKDIV 置为逻辑“1”, 用户可以在 CLK_I 和 CLK_O 之间用一个 4.9152 MHz 的晶体, 而在器件内部用规定的 2.4576 MHz 进行操作。CLKDIV 置为逻辑 0 时, CLK_I 引脚处的频率实际上就是器件内部的频率。

CLK: 时钟位。CLK 应根据 AD7705/06 的工作频率而设置。如果转换器的主时钟频率为 2.4576 MHz($\text{CLKDIV} = 0$)或 4.9152 MHz($\text{CLKDIV} = 1$), 则 CLK 应置“0”。如果器件的主时钟频率为 1 MHz($\text{CLKDIV} = 0$)或 2 MHz($\text{CLKDIV} = 1$), 则该位应置“1”。

FS1、FS0: 滤波器选择位。FS1、FS0 与 CLK 一起决定器件的输出更新率。表 3-13 给出了滤波器的输出更新率和在 -3 dB 处的截止频率。

表 3-13 滤波器输出更新率和在-3 dB 处的截止频率

CLK	FS1	FS0	输出更新率/Hz	滤波器在-3 dB 处的截止频率/Hz
0	0	0	20	5.24
0	0	1	25	6.55
0	1	0	100	26.2
0	1	1	200	52.4
1	0	0	50	13.1
1	0	1	60	15.7
1	1	0	250	65.5
1	1	1	500	131

(4) 测试寄存器(RS2、RS1、RS0 = 100)。测试寄存器用于测试器件时, 建议用户不要改变测试寄存器的默认值(上电或复位时自动置入全 0), 否则当器件处于测试模式时, 不能正确运行。

(5) 零标度校准寄存器(RS2、RS1、RS0 = 110)。AD7705/06 包含几组独立的零标度寄存器, 每个零标度寄存器负责一个输入通道。它们皆为 24 位读/写寄存器, 24 位数据必须被写之后才能传送到零标度校准寄存器。零标度寄存器和满标度寄存器连在一起使用, 组成一个寄存器对。每个寄存器对对应一对通道。当器件被设置成允许通过数字接口访问这些寄存器时, 器件本身不再访问寄存器系数以使输出数据具有正确的结果。在访问校准零标度校准寄存器(无论是读/写操作)后, 从器件读得的第一个输出数据可能包含了不正确的数据。此外, 数据校准期间, 校准寄存器不能进行写操作。这类操作可以通过以下方法避免: 在校准寄存器开始工作前, 将模式寄存器的 FSYNC 位置为高电平, 任务结束后, 又将其置为低电平。

(6) 满标度校准寄存器(RS2、RS1、RS0 = 111)。AD7705/06 包含几个独立的满标度寄存器, 每个满标度寄存器负责一个输入通道。它们皆为 24 位读/写寄存器, 24 位数据必须被写之后才能传送到满标度校准寄存器。其方法同零标度校准寄存器的编程。

2) 校准过程

AD7705/06 包括自校准、零标度系统校准和满标度系统校准三种校准类型。有两种方法可用来判断校准是否结束。第一种方法是: 监视 $\overline{\text{DRDY}}$, 若 $\overline{\text{DRDY}}$ 返回低电平, 则说明校准过程已经结束, 同时也表明数据寄存器中有一个新的有效数据, 这一新的数据就是校准结束后一次正常的转换结果。第二种方法是: 监视设置寄存器的 MD1、MD0 位, 若 MD1、MD0 回到“0”(校准后, MD1、MD0 返回“0”), 则表明校准过程已经结束, 这种方法不能判断数据寄存器中是否有新的转换结果, 但它比第一种判断方法在时间上要早, 也就是能更快地知道校准是否结束。 $\overline{\text{DRDY}}$ 回到低电平的过程包括一次正常的转换过程和使第一次转换结果延迟的过程。

3) 模拟输入

(1) 模拟输入范围。AD7705 包括 2 个模拟输入对, 即 IN1+、IN1- 和 IN2+、IN2-。模拟输入对提供可编程增益, 可处理单、双极性输入信号。应注意, 双极性输入信号以各自的 IN-端为参考。AD7706 包括 3 个伪差模拟输入对, 即 IN1、IN2 和 IN3, 这些输入以器件的 COM 端为参考。

在非缓冲模式下,共模输入范围为 GND 到 Vcc,这就表明器件可以处理所有增益的单、双极性输入信号。25℃时,在不使性能下降的情况下,模拟输入可以达到绝对电压 GND~200 mV,但漏电流随温度上升而显著增大,绝对输入电压范围被限制在 GND + 50 mV~Vcc + 30 mV 之间,它还限制共模输入范围。这就是说,在缓冲模式下,双极性输入范围的允许增益要受到限制。因此必须仔细设置共模电压和电源范围,以确保它们不超出上述极限,否则,器件的线性度将会降低。

(2) 输入采样率。AD7705/06 调制器的采样频率维持在 $f_{CLK}/128$ (19.2 kHz 时, $f_{CLK} = 2.4576$ MHz),而与增益选择无关,但是,大于 1 的增益是通过在每个调制器周期中多重输入采样以及基准电容与输入电容之比的倍数得到的。作为多重采样的结果,输入采样率随选定的增益而变化。

(3) 单极性/双极性输入。无论是单极性还是双极性电压,AD7705/06 的模拟输入端都能接收。双极性输入并不表示器件能够处理模拟输入端的负电压,因为模拟输入电压不能小于 -30 mV,以确保器件的正常工作。输入通道是全差分的。因此,对于 AD7705,IN+ 输入电压以各自的 IN- 为基准;对于 AD7706,加到模拟输入通道的电压以 COM 为基准。选择单极性还是双极性输入是由设置寄存器的 B/U 位来决定的。无论是单极性还是双极性输入,都不改变任何输入信号的状态,只改变输出数据的代码和转换函数上的校准点。

(4) 基准输入。REF+ 和 REF- 为 AD7705/06 提供差分基准输入功能,差分输入的共模范围是 GND~Vcc。当 AD7705/06 以 5 V 电源电压工作时,基准电压为 +2.5 V;电源电压为 3 V 时,基准电压为 +1.225 V。为确保器件能够准确无误地工作,必须使 REF+ 大于 REF-。

4) 数字与模拟滤波

AD7705/06 包含一个片内低通数字滤波器,可用来处理器件的 $\Sigma\Delta$ 调制器的输出信号,所以,该器件不仅提供模/数转换功能,还具备一定的滤波能力。

数字滤波与模拟滤波存在许多系统差异。

一方面,数字滤波发生在模/数转换之后,它能消除模/数转换过程中产生的噪声,而模拟滤波不能做到这一点。此外,数字滤波比模拟滤波更容易实现可编程性。

另一方面,在模拟信号进入 ADC 之前,模拟滤波能够消除重叠在模拟信号上的噪声,数字滤波则不能做到这一点,并且寄生在信号上的噪声峰值接近满标度时,即使信号的平均值在极限范围内也有可能使模拟调制器和数字滤波器达到饱和状态。为了解决这个问题,在 AD7705/06 的 $\Sigma\Delta$ 调制器和数字滤波器内部建立了一个峰值处理,这允许超出模拟输入范围的 5%。若噪声信号比这还要大,那么就考虑输入端的模拟滤波,或降低输入通道电压,使输入电压的范围为模拟输入通道电压满标度范围的一半。这样动态范围降低 5%,将使超范围性能增加 1 倍。

(1) 后置数字滤波。当 f_{CLK} 为 2.4576 MHz 时,片上调制器提供 19.2 kHz 的采样输出速率。片内的数字滤波器对这些采样速率进行取样后,提供一定输出速率的数据。因为输出速率比 Nyquist 标准要高,所以对于给定了带宽和噪声性能的应用来说,这一输出速率能满足大多数应用要求。但对于有些在给定带宽和噪声性能时需要更高输出速率的特殊应用来说,则要在 AD7705/06 的数字滤波器之后配置后置滤波功能。例如,若带宽要求为 7.86 Hz,而所需的更新率是 100 Hz,则当数据的输出速率为 100 Hz 时给出的 -3 dB 带宽是 26.2 Hz。后置滤波可以应用在这种场合,它可以将带宽减小到 7.86 Hz,同时减小输出噪声,而保持

输出率仍为 100 Hz。后置滤波还可以用来降低带宽小于 13.1 Hz 的器件产生的输出噪声,增益为 128,带宽为 13.1 Hz 时,输出噪声的均方根值是 450 nV。这是器件主要的噪声,即白噪声,并且由于输入被削波,因而噪声具有平坦的频率响应。通过将带宽降低到低于 13.1 Hz,在最终的通带内的噪声将减小。带宽以 2 的倍数降低将导致输出噪声以近于 1.25 的倍数减小。这一附加的后置滤波使得稳定时间变得更长。

(2) 模拟滤波。前面已提到,对调制器采样频率的整数倍,数字滤波器不能起到抑制作用。但是,因为 AD7705/06 具有超采样率,所以这些波段只占整个频谱的一小部分,大部分宽频噪声都被滤掉了。这就是说,与没有片内滤波的传统转换器相比,AD7705/06 的前端模拟滤波要求已大大降低。此外,由于该器件的 100 dB 的共模抑制已达到数千赫兹,因此这一频段范围内的噪声将大大降低。但是在具体应用中,可能需要从数字滤波器能通过的频段上消除不需要的频率,需要在 AD7705/06 的前端加上衰减功能。在另外一些应用中,可能要在 AD7705/06 的前端进行模拟滤波,以免有用频带外的差分噪声信号使模拟调制达到饱和。

在非缓冲模式下,如果在 AD7705/06 的前端有无源元件,则必须确保电源阻抗足够低,以免在系统中引入增益误差。这极大地限制了 AD7705/06 前端无源反混叠滤波在非缓冲模式下的使用。但是当器件在缓冲模式下工作时,大电源电阻只会产生一个很小的直流偏移误差(10 k Ω 电源电阻引起不到 10 μ V 的偏差误差)。因此,如果系统需要在 AD7705/06 前端使用无源模拟滤波,则建议器件在缓冲模式下工作。

5) 校准

AD7705/06 提供了多种校准选择,具体选择哪种校准可以由设置寄存器的 MD1 和 MD0 位来编程。一旦给 MD1 和 MD0 位写入数据,一个校准周期就开始了。当工作环境温度和电压发生变化时,应对器件进行校准;当选定的增益、滤波器陷波或单极性/双极性输入范围发生变化时,也应进行校准。

校准分为自校准和系统校准。对选定的通道进行全域校准时,片上微控制器必须在两种不同的输入状态下记录调制器的输出,也就是“零标度”和“满标度”点。这些点是在校准过程中,在调制器的输入端输入不同的电压值后,器件执行一次转换而得到的结果。当然,校准精度也只能和正常模式下提供的噪声水平相当。零标度校准转换的结果存储在零标度校准寄存器中,而满标度校准转换的结果存储在满标度校准寄存器中。依靠这些数据,微控制器就能计算出转换器的输入/输出转换函数的偏移和增益斜率,器件以 33 位分辨率来确定 16 位转换结果。

(1) 自校准。通过向设置寄存器的 MD1 和 MD0 写入相应值(0, 1),器件开始自校准。在单极性输入信号范围内,用来确定校准系数的零标度点用差分输入对的输入端在器件内部短路(即对于 AD7705, $(IN+) = (IN-) =$ 内部偏置电压;对于 AD7706, $IN = COM =$ 内部偏置电压)。增益可编程放大器(PGA)设置为用于零标度校准转换时选定的增益(由通信寄存器内的 G2~G0 位设置)。满标度标准转换是在一个内部产生的基准电压和选定增益的条件下完成的。

对于双极性输入范围的自校准,整个过程与上述过程相似,零标度和满标度点几乎与单极性输入的一样,但由于 AD7705/06 是配置成双极性输入工作的,因而使输入范围缩小了。

(2) 系统校准。通过系统校准, AD7705/06 可以对系统增益、偏移误差以及器件本身的内部误差进行补偿。系统校准采用和自校准一样的斜率系数进行计算, 但所用电压是系统对 IN 输入端用于零和满标度校准的电压值。

在单极性模式下, 系统校准在转换函数的两个端点之间完成; 在双极性模式下, 它在中标度(零差分电压)和正的满标度之间完成。

系统校准是分两步进行的, 在全系统的校准序列完成之后, 偏移和增益校准能自动执行, 以调节系统零基准点或系统增益。校准系统偏移和增益两个参数中的任何一个, 不会影响另一个。

当器件在非缓冲模式下使用时, 系统校准还可以用来消除模拟输入端由电源阻抗引入的误差。模拟前端一个简单的 R、C 反重叠滤波器就可能给模拟输入电压引入增益误差, 但是系统校准可以消除这种误差。

4. AD7705/06 的使用

1) 时钟和振荡器电路

AD7705/06 要求外部主时钟输入, 这个主时钟输入可以是 CLK_o 脚不连接时加在 CLK_i 引脚上的一个外部时钟信号, 或者在 CLK_i 和 CLK_o 两个引脚之间连接的一个频率合适的晶体或陶瓷谐振器。在此情况下, 时钟电路作为振荡器工作, 为 AD7705/06 提供主时钟信号。主时钟频率 f_{CLK} 直接影响输入采样频率、调制器采样频率、-3 dB 频率、输出更新率和校准时间。

在 CLK_i 和 CLK_o 两个引脚之间配置一个晶体或陶瓷谐振器比采用在 CLK_i 引脚处引入驱动时钟信号的工作电流大。这是因为片内振荡电路在使用晶体或陶瓷谐振器的情况下更活跃。因此, 在 CLK_i 引脚处施加一个外部时钟, 而将 CLK_o 悬空时, 可使 AD7705/06 的功耗降低。

振荡器所消耗的额外电流的大小取决于很多因素, 连接 CLK_i 和 CLK_o 两个引脚间的电容器的容量越大, 消耗电流越大。注意不能超过晶体或陶瓷谐振器厂商推荐的电容值, 这些值一般在 30~50 pF 范围内。

在振荡电路开始振荡之前, 它还需要一个启动过程。当 $V_{CC} = 5\text{ V}$ 时, 晶体振荡器的频率为 4.9512 MHz、2.4576 MHz 和 1 MHz, 所对应的启动时间分别是 6 ms、16 ms 和 20 ms。 V_{CC} 降为 3 V 时, 在相同频率条件下, 启动时间缩短 20%。

AD7705/06 的主时钟可以从 CLK_o 引脚引出, 加在此引脚上的最大推荐负载为一个 CMOS 负载。当用晶体或陶瓷谐振器产生时钟信号时, 可能需要把这个时钟作为系统的时钟源。在这种情况下, 建议用 CMOS 缓冲器对 CLK_o 信号在加到系统电路之前进行缓冲。

2) 系统同步

设置寄存器中的 FSYNC 允许用户在不影响 AD7705/06 设置状态的情况下, 对调制器和数字滤波器进行复位。FSYNC 置 1 时, 数字滤波器和模拟调制器处于已知复位状态, 此时 AD7705/06 不处理任何输入采样。当置 0 时, 调制器和滤波器不再处于复位状态, AD7705/06 又开始从下一个时钟沿收集采样。

FSYNC 输入也可以用作允许器件在常规变换模式下工作的软件启动转换命令。在这种模式下, 数据写入 FSYNC, 转换开始, \overline{DRDY} 下降沿提示转换完成。这一方案的缺点在于:

每一个数据寄存器的数据更新都得考虑滤波器的稳定时间,因此,数据寄存器的更新时间要多 3 倍。

由于 FSYNC 对数字滤波器进行复位,因此在新字写入输出寄存器前,整个稳定时间必须结束。当 FSYNC 为 0 时, $\overline{\text{DRDY}}$ 处于低电平, FSYNC 命令将不对 $\overline{\text{DRDY}}$ 复位,使其变为高电平。这是因为数据寄存器中有一个还未读的数,在数据寄存器进行数据更新前, $\overline{\text{DRDY}}$ 线将保持低电平,同时停留在低电平直到数据寄存器发生更新。从数据寄存器读数据将使 $\overline{\text{DRDY}}$ 信号变高,直到滤波器的稳定时间已消逝(从 FSYNC 命令后)并且在数据寄存器中已有一个有效字, $\overline{\text{DRDY}}$ 才回到低电平。如果当 FSYNC 命令已发出时 $\overline{\text{DRDY}}$ 为高,则 $\overline{\text{DRDY}}$ 在滤波器的稳定时间消逝后才能回到低电平。

3) 复位输入

复位输入电路复位所有的逻辑、数字滤波器和模拟调制器,同时将所有的片内寄存器设置到其默认状态。当 RESET 输入信号处于低电平时, $\overline{\text{DRDY}}$ 处于高电平, AD7705/06 忽略发往寄存器的任何通信数据。当 RESET 返回高电平时,器件才开始处理数据。

即使 RESET 输入处于低电平,片内振荡器电路仍继续工作,CLKo 引脚的主时钟信号继续有效。因此,在由 AD7705/06 提供系统时钟的应用中,AD7705/06 在复位过程中将产生一个不间断的主时钟信号。

4) 等待模式

在不需要提供转换结果的情况下,通信寄存器中的 STBY 位允许用户将器件设置在掉电模式下工作。在等待模式下,AD7705/06 保留所有片内寄存器(包括数据寄存器)中的所有内容。脱离等待模式后,器件开始处理数据,在 STBY 位写入 0 的 3x1/输出速率时间后,数据寄存器中可以有新的有效数据。

STBY 位不影响数字接口,也不影响 $\overline{\text{DRDY}}$ 位的状态。如果 $\overline{\text{DRDY}}$ 处于高电平,而 STBY 处于低电平,则它将保持高电平直到数据寄存器中有新的有效字。如果 $\overline{\text{DRDY}}$ 处于低电平,STBY 也处于低电平,则它将保持低电平直到数据寄存器被更新。如果在 $\overline{\text{DRDY}}$ 为低电平时,器件进行等待模式(表明数据寄存器中有未读的有效字),则可以在等待模式下读出数据寄存器中的数据。读操作后, $\overline{\text{DRDY}}$ 将回到高电平。

5) 接地及布线

由于模拟输入和基准输入是差分的,因此模拟调制器的大部分电压都是共模电压。AD7705/06 良好的共模抑制性能可消除模拟输入信号里的共模噪声。数字滤波器能抑制供电电源产生的宽带噪声。此外,数字滤波器还能消除基准输入信号里的噪声。总之,AD7705/06 比传统的高分辨率的 A/D 转换器更能消除噪声的干扰。但是,由于它的分辨率太高,而要求噪声电平太小,因此必须注意接地和器件电路布线。

AD7705/06 的印制板电路必须严格设计,以确保模拟区和数字区分开,并各自限定在电路板上一定区域内。利用接地平面可以很容易地将它们分开。最好用腐蚀技术做接地平面,因为这样能使屏蔽性能最好。应只在一个地方将模拟和数字接地平面连接在一起,以避免出现接地环路。

应避免在器件下面走数字线,否则会造成片内噪声成倍增加。模拟接地平面应布在器件下面。AD7705/06 的电源线应足够粗以便降低线路阻抗,同时减少电源供电线路的峰值信号的影响。主时钟类的快速跳变信号应用数字接地屏蔽,以免将噪声辐射到电路的其他

部分。时钟信号不能在模拟输入信号附近通过。模拟信号和数字信号之间应避免相互交叉。电路板两面的线路应走成直角,这样可以降低电路板的馈通效应。采用微带线技术最好,不过并不能总是使用双面电路板。应用这项技术时,电路板上元件的一边放在接地平面上,信号则放在电路板上焊接的一边。

使用高分辨率的 ADC 时,良好的去耦性能很重要。所有的模拟电源都应去耦。方法是:用 $10\ \mu\text{F}$ 并联一个 $0.1\ \mu\text{F}$ 的陶瓷电容器接 GND 去耦。为使去耦元件获得最佳效果,应使它们尽量靠近 ADC,在 ADC 的正上方最为理想。所有逻辑芯片都应用一个连在数字 GND 上的 $0.1\ \mu\text{F}$ 的电容去耦。

6) 数字接口

AD7705/06 的编程功能通过片内寄存器的设置来控制。对这些寄存器的“读/写”操作通过器件的串行接口来完成。

AD7705/06 的串行接口包括 5 个信号,即 $\overline{\text{CS}}$ 、SCLK、DIN、DOUT 和 $\overline{\text{DRDY}}$ 。DIN 线用来向片内寄存器传输数据,而 DOUT 线用来访问寄存器里的数据。SCLK 是串行时钟输入,所有的数据传输都和 SCLK 信号有关。 $\overline{\text{DRDY}}$ 线作为状态信号,用以提示数据什么时候已准备好从寄存器读数据。输出寄存器中有新的数据字时, $\overline{\text{DRDY}}$ 变为低电平。在输出寄存器数据更新前,若 $\overline{\text{DRDY}}$ 变为高电平,则提示这个时候不读数据,以免在寄存器更新的过程中读数据。 $\overline{\text{CS}}$ 用来选择器件,在有许多器件与串行总线相连的应用中,它也用于对系统中的 AD7705/06 进行解码。

图 3-13 是用 $\overline{\text{CS}}$ 对 AD7705/06 进行解码的时序图。其中图 3-13(a)为 AD7705/06 的输出移位寄存器读数据的时序图,而图 3-13(b)是向输入移位寄存器写入数据的时序图。即使是在第一次读操作后 $\overline{\text{DRDY}}$ 线返回高电平,也可能出现两次从输出寄存器读到同样数据的情况。必须注意确保在下次输出更新数据之前,读操作已经完成。

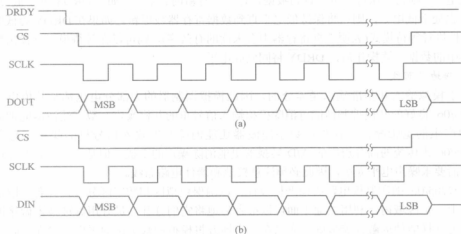


图 3-13 AD7705/06 的读/写时序

(a) 读周期时序图; (b) 写周期时序图

AD7705/06 也可以在 $\overline{\text{CS}}$ 被用作帧同步信号时工作。这种方案适合于与 DSP 接口,在这

种情况下, 首位(MSB)被 $\overline{\text{CS}}$ 时序有效输出, 因为 $\overline{\text{CS}}$ 通常是在 DSP 上的 SCLK 处于下降沿时产生的。假如时序不变更, SCLK 也可在两次相邻的数据传输间继续运行。通过加在 AD7705/06 的 RESET 脚上的复位信号能够复位串行接口, 还能够通过向 DIN 输入端写入一系列的“1”来复位串行接口。如果在至少 32 个串行时钟周期内向 AD7705/06 的 DIN 线写入逻辑“1”, 则串行接口就被复位。这保证了在 3 线系统中, 由于软件错误或系统中的闪烁信号造成接口迷失, 使接口回到 AD7705/06 等待对其通信寄存器进行一次写操作的状态。这一写操作本身并不复位任何寄存器的内容, 但因为接口已经迷失, 写入任何寄存器的信息都是未知的, 所以建议对所有的寄存器重新设置一次。

有一些微处理器或微控制器的串行接口只有一根单独的串行数据线。在这种情况下, 可以把 AD7705/06 的 DOUT 和 DIN 线连接在一起并把它们与处理器的单根数据线相连。在这根单一的数据线上必须使用一个 10 k Ω 的上拉电阻。在这种情况下, 如果接口迷失, 则因为读、写操作共享同一根线, 所以复位并使接口还原到已知状态的过程与以前叙述的有所不同。这一过程要求 24 个时钟的读操作和至少 32 个连续时钟周期的逻辑“1”的写操作, 以保证串行接口回到已知状态。

5. 典型的微机接口

AD7705/06 中的大多数寄存器都是 8 位寄存器, 这使得与带有 8 位串行接口的微控制器连接非常容易。AD7705/06 上的数字寄存器为 16 位, 偏移和增益寄存器为 24 位, 向这些寄存器和微控制器端口传输多个 8 位字节, DSP 处理器和微处理器通常在串行数据操作中传输 16 位数据。有些处理器(如 ADSP-2105)在一次串行数据传输中可编制周期数。这就允许用户在任何传输中可增减寄存器的位数, 使之与要求相匹配。尽管 AD7705/06 中有些寄存器只有 8 位, 但可以将两个这样的寄存器拼起来, 写操作就可以作为一个 16 位数据传输处理。例如, 如果设置寄存器要被更新, 则处理器必须首先对通信寄存器进行写操作, 然后写一个 8 位数据到设置寄存器。如果需要, 这些可以由单次的 16 位数据传输来完成, 因为写到通信寄存器中的 8 位串行数据一旦完成, 器件立即将自己设置成对设置寄存器进行了一次写操作的状态。

1) AD7705/06 与 68HC11 接口

图 3-14(a)为 AD7705/06 与 68HC11 微控制器接口的连接图。这种方案采用 3 线接口, 而将 AD7705/06 的 CS 线连接到低电平。在这个方案中, DRDY 位被监控以决定数据何时被更新, 这要将接口线增加到四条。对 DRDY 线监控可以通过两种方法: 第一种就是将 DRDY 与 68HC11 的其中一个输入端口位相连(例如 PC0), 此端口位被查询并决定 DRDY 线的状态; 另一种方法是使用一个中断线, 在这种情况下, DRDY 与 68HC11 的 IRQ 输入线相连。对于要求控制 $\overline{\text{CS}}$ 输入线的接口, 可将 68HC11 的其中一个端口(例如 PC1)配置成输出口, 用来驱动 $\overline{\text{CS}}$ 输入。

68HC11 配置成主机模式且 CPOL 位置逻辑 1, CPHA 位置逻辑 1。这样配置使 SCLK 线在两次数据传输之间为高电平。AD7705/06 不能全双工工作。当 AD7705/06 被配置成写操作时, 即使 SCLK 输入有效, DOUT 线上也不出现数据。同理, 若器件被配置成读操作, 则即使 SCLK 输入有效, 出现在 DIN 线上的数据仍被忽略。

在此例中, $\overline{\text{DRDY}}$ 输出线与 68HC11 的 PC0 端口相连, 进行查询以决定它的状态。

2) AD7705/06 与 8xC51 接口

AD7705/06 与 8xC51 微控制器的接口电路如图 3-14(b)所示。图中,在将 $\overline{\text{CS}}$ 接低电平的情况下,采用 2 线连接。 $\overline{\text{DRDY}}$ 位被监控以决定何时数据寄存器被更新。另一种方案是监控 $\overline{\text{DRDY}}$ 输出线,这要将接口线增加到 3 线。对 $\overline{\text{DRDY}}$ 线的监控方法有 2 种:即查询方法和中断方法,类似于 68HC11 的接口方法。8xC51 配置为串行接口方式 0 模式,这种串行接口包括单一的一根数据线的。其结果是,AD7705/06 的 DOUT 和 DIN 引脚必须连接在一起,还必须连接一个 $10\text{ k}\Omega$ 的上拉电阻。

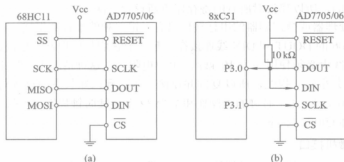


图 3-14 AD7705/06 与微机的接口

(a) AD7705/06 与 68HC11 接口; (b) AD7705/06 与 8xC51 接口

3.3.2 应用电路与编程

AD7705 具有双通道,低成本,分辨率高等特点。由于采用 $\Sigma\Delta$ 结构实现模/数转换,因此该器件在噪音环境下能免受干扰。同时它还提供了可编程的增益放大、数字滤波器和校准功能。因此,它比普通的积分式 ADC 在抗干扰方面性能更优越。

片内 PGA 允许 AD7705 处理低到 10 mV (满标度)的模拟输入电压(基准电压为 $+1.25\text{ V}$)。器件在非缓冲模式下工作时,差分输入使模拟输入范围的绝对值处于 $\text{GND}\sim\text{Vcc}$ 之间的任一值。由于允许用户将传感器直接与 AD7705 的输入端相连,因此 AD7705 的可编程增益前端允许处理 $+20\text{ mV}\sim+2.5\text{ V}$ 之间的单极性模拟输入信号和 $\pm 20\text{ mV}\sim\pm 2.5\text{ V}$ 的双极性信号。

AD7705 的一个典型应用就是压力测量。图 3-15 是 AD7705 与一个压力传感器直接相连的情况。压力传感器被连接成桥式电路,输出差分电压。当在传感器上加上满标度压力 (300 mmHg)时,差分输出电压(即 IN^+ 和 IN^- 两端之间的电压)是输入电压的 3 mV/V 。假定电桥激励电压是 $+5\text{ V}$,则传感器的满标度输出电压是 15 mV 。桥式电路的激励电压还用来为 AD7705 产生基准电压。因此,激励电压的变化不会造成系统内的误差。图 3-15 中当两个电阻值分别为 $24\text{ k}\Omega$ 和 $15\text{ k}\Omega$,激励电压为 5 V 时,AD7705 产生的基准电压为 1.92 V 。器件具有 128 的可编程增益时,AD7705 的满标度输入幅度应是 15 mV ,此值与传感器的输出范围有关。AD7705 的第二个通道可作为一个辅助通道以测量另一个变化(如温度),见图 3-15 中的热电偶。这个次级通道可以用来调整初次通道的输出信号,以便消除温度对系统的影响。

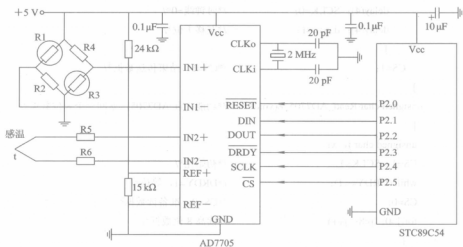


图 3-15 AD7705 的压力测量电路

采样压力通道的 C51 源程序如下:

```
#include "reg52.h"

#define SAMPLES 1000 /*采样延时常数*/

sbit RESET = P2^0; /*定义复位引脚*/
sbit DIN = P2^1; /*定义输入引脚*/
sbit DOUT = P2^2; /*定义输出引脚*/
sbit CS = P2^5; /*定义片选引脚*/
sbit SCLK = P2^4; /*定义时钟引脚*/
sbit DRDY = P2^3; /*定义 DRDY 引脚*/

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

void WriteAD7705(unsigned char byte) /*向 AD7705 写入一个字节*/
{
    unsigned char dat, j; /*分别用于传递形参*/
    dat=byte; /*发送数据*/
    CS=1; SCLK=1; /*初始化*/
    CS=0; /*CS=0, 准备开始送数*/
    for(j=0; j<8; j++)
    {
        if(dat&0x80) DIN=1;
        else DIN=0;
        delay(4); SCLK=1; /*时钟线=1*/
    }
}
```



```

        delay(4); SCLK=0;          /*时钟线=0*/
        delay(4); dat<<=1;        /*左移 1 位*/
    }
    CS=1;                          /*CS=1, 结束传送数据*/
}

unsigned char Read_AD7705_8(void) /*功能: 读 AD7705, 返回一个 8 位数据*/
{
    unsigned char j, x;
    CS=1; SCLK=1;                  /*初始化*/
    while(DRDY==1);                /*DRDY=1, 等待*/
    CS=0;                           /*CS=0, 准备读数据*/
    for(j=0; j<8; j++)             /*读高 8 位数据*/
    {
        x<<=1; delay(4);
        SCLK=1; delay(4);
        if(DIN==0)    x |= 0xfe;
        else    x |= 0x01;
        SCLK=0;        /*时钟线置低*/
    }
    CS=1;                  /*CS=1, 读数据结束*/
    return(x);             /*返回 8 位数据*/
}

unsigned int Read_AD7705_16(void) /*功能: 读 AD7705, 返回值是整型数据*/
{
    unsigned int x;
    unsigned char j;
    CS=1; SCLK=1;          /*初始化*/
    while(DRDY==1);        /*DRDY=1, 等待*/
    CS=0;                   /*CS=0, 准备读数据*/
    for(j=0; j<16; j++)    /*读 16 位数据*/
    {
        x<<=1; delay(4);
        SCLK=1; delay(4);
        if(DIN==0)    x |= 0xfffe;
        else    x |= 0x0001;
        SCLK=0;
    }
    CS=1;                  /*CS=1, 读数据结束*/
    return(x);             /*返回 A/D 的(16 位数据)值*/
}

```

```

    }
void main (void)
{ unsigned int i;
  unsigned int ad_data;
  RESET=0;          /*复位*/
  delay(200);
  RESET=1;
  CS=1;             /*CS=1(高电平)*/
  WriteAD7705(0x20); /*设置1通道(IN1+, IN1-), 下一操作是设置时钟寄存器*/
  WriteAD7705(0x00); /*设置主频为2MHz, 输出更新速率为20Hz*/
  WriteAD7705(0x10); /*设置1通道(IN1+, IN1-), 下一操作是设置寄存器*/
  WriteAD7705(0x40); /*设置增益为1, 自校准模式*/
  while(DRDY==1);   /*等待 DRDY=0*/
  for(i=0; i<SAMPLES; i++); /*等待采样*/
  while(1)           /*对于具体的应用应当改写该部分*/
  {
    WriteAD7705(0x38); /*设置1通道(IN1+, IN1-), 下一操作是读数据寄存器的值*/
    ad_data=Read_AD7705_16(); /*读16位A/D的值*/
    delay(100);
  }
}

```

3.4 AD7714 高性能 24 位 A/D 转换器

3.4.1 硬件与功能描述

高性能的 AD7714 可直接把来自传感器的小信号直接转换成数字量输出。它使用和-差转换技术以实现高达 24 位的无误差性能。通过片内控制寄存器可对数字滤波器的第 1 个凹口编程, 允许调整滤波器的截止频率和稳定时间。3 个标准的差分模拟输入也可以配置为 5 个准差分模拟输入, 对每个通道可实现信号的调理和转换。该器件采用单电源(+3 V 或+5 V)工作和串行接口。

AD7714 可理想地用于便携式工业仪表、便携式重量计、循环供电系统及压力传感器等方面。

1. 主要性能特点

- (1) A/D 精度为 24 位无误差输出;
- (2) 非线性: <0.0013%;
- (3) 增益在 1~128 范围内可调;
- (4) 掉电电流小于 5 μ A;

- (5) 3个标准的差分输入,可配成5个准差分模拟输入;
- (6) 差分基准输入;
- (7) 内部提供数字滤波;
- (8) 提供SPI、QSPI串行接口;
- (9) 只需2.7~3.3V或4.75~5.25V单电源;
- (10) CMOS结构确保非常低的功耗,掉电模式时功耗减少到15 μW (典型值);
- (11) 工作温度范围为-40~+105 $^{\circ}\text{C}$ 。

2. 内部结构与引脚说明

AD7714的内部结构与引脚排列如图3-16所示。其引脚功能见表3-14。

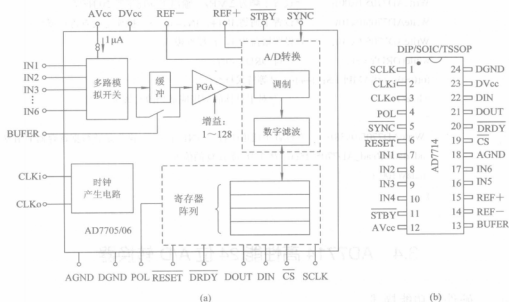


图3-16 AD7714的内部结构与引脚排列

(a) 内部结构; (b) DIP/SOIC/TSSOP 引脚排列

表3-14 AD7714的引脚功能

引脚	名称	功能描述
1	SCLK	串行时钟输入信号
2, 3	CLKi, CLKo	时钟信号输入/输出(一般接晶振), 一般为1~3 MHz
4	POL	时钟极性。该引脚为“1”时表示闲置时钟为“1”; 否则, 表示闲置时钟为“0”
5	SYNC	当使用多个AD7714时, 它用于数字滤波器和模拟调制器的同步。该引脚为“0”时具有复位功能, 该引脚为“1”时不影响接口传送数据
6	RESET	复位输入, 低电平有效
7	IN1	模拟输入通道1, 可编程模拟输入端。当与IN6一起使用时, 可用作准差分输入端; 当与IN2一起使用时, 可用作差分模拟输入的正输入端

续表

引脚	名称	功能描述
8	IN2	模拟输入通道 2, 可编程模拟输入端。当与 IN6 一起使用时, 可用作准差分输入端; 当与 IN1 一起使用时, 可用作差分模拟输入的负输入端
9	IN3	模拟输入通道 3, 可编程模拟输入端。当与 IN6 一起使用时, 可用作准差分输入端; 当与 IN4 一起使用时, 可用作差分模拟输入的正输入端
10	IN4	模拟输入通道 4, 可编程模拟输入端。当与 IN6 一起使用时, 可用作准差分输入端; 当与 IN3 一起使用时, 可用作差分模拟输入的负输入端
11	$\overline{\text{STBY}}$	逻辑输入端。该引脚为“0”时, 将关断模拟和数字电路, 把电流消耗减少到 5 μA (典型值)
12	AVcc	模拟正电源输入端。该引脚接+3.3 V 或+5 V
13	BUFER	缓冲器选项。当输入为低电平时, 模拟输入不用内部缓冲; 否则用内部缓冲器, 以提高模拟输入阻抗
14	REF-	基准“-”输入端
15	REF+	基准“+”输入端
16	IN5	模拟输入通道 5, 可编程模拟输入端。当与 IN6 一起使用时, 可用作差分模拟输入的正输入端
17	IN6	模拟输入通道 6。在准差分模式下, 它是 IN1~IN4 的基准点。当与 IN5 一起使用时, 它用作差分模拟输入的负输入端
18	AGND	模拟参考地
19	$\overline{\text{CS}}$	片选, 低电平有效
20	$\overline{\text{DRDY}}$	逻辑输出, 可以获取新的输出字。该引脚可用来指示何时完成校准
21	DOUT	串行数据输出端
22	DIN	串行数据输入端
23	DVcc	数字正电源输入端。该引脚接+3.3 V 或+5 V
24	DGND	数字参考地

3. 内部寄存器说明

AD7714 包含 8 个片内寄存器, 它们可通过器件的串行口访问。第 1 个寄存器是通信寄存器, 它控制通道的选择, 决定下一个操作是读或写操作, 还决定下一个读或写操作访问哪一个寄存器。第 2 个寄存器是模式寄存器, 它决定校准模式和增益的设置。第 3 个寄存器称为滤波器高 8 位, 它决定字的长度、双极性/单极性操作并包括滤波器选择字的高 4 位。第 4 个寄存器称为滤波器低位 8 位。第 5 个寄存器是测试寄存器, 它在测试器件时被访问。第 6 个寄存器是数据寄存器, 从这个寄存器可访问器件的输出数据。最后一个寄存器是可访问的校准寄存器。零刻度校准寄存器允许访问所选输入通道的零刻度校准系数, 而满刻度校准寄存器允许访问所选输入通道的满刻度校准系数。

1) 通信寄存器(RS2~RS0=000)

通信寄存器是 8 位寄存器, 从该寄存器可以读出数据或把数据写入此寄存器。与器件的所有通信必须从对通信寄存器的写操作开始。写至通信寄存器的数据决定下一个操作是读操作或写操作并决定对哪一个寄存器发生此操作。一旦完成后续的对所选择寄存器的读

或写操作, 接口便返回到等待对通信寄存器写操作的状态。这是接口的缺省状态, 在上电时或 RESET 后, AD7714 处于此缺省状态, 等待对通信寄存器的写操作。在失掉接口序列的情况下如果发生足够时间宽度(至少包括 32 个串行时钟)的写操作, 且 DIN 为高电平, 那么 AD7714 将返回到此缺省状态。通信寄存器的格式如表 3-15 所示。

表 3-15 通信寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
0/ $\overline{\text{DRDY}}$	RS2	RS1	RS0	R/ $\overline{\text{W}}$	CH2	CH1	CH0

其中:

0/ $\overline{\text{DRDY}}$: 对于写操作, 必须把 0 写入该位以便对通信寄存器写操作。如果把 1 写入此位, 那么器件将不能操作后续的位, 它将停留在这个位的地址直到把 0 写到这一位为止。一旦把 0 写入此位, 接着的 7 位将装入通信寄存器。对于读操作, 这一位提供器件的 DRDY 标志状态。这一位的状态与 $\overline{\text{DRDY}}$ 输出引脚相同。

RS2~RS0: 寄存器选择位。这 3 个位决定 8 个片内寄存器中哪一个进行下一次(紧跟着)的读或写操作。寄存器选择见表 3-16。

表 3-16 寄存器选择

RS2	RS1	RS0	寄存器名	寄存器位数
0	0	0	通信寄存器	8 位
0	0	1	模式寄存器	8 位
0	1	0	滤波器高 8 位寄存器	8 位
0	1	1	滤波器低 8 位寄存器	8 位
1	0	0	测试寄存器	8 位
1	0	1	数据寄存器	16 位或 24 位
1	1	0	零刻度校准寄存器	24 位
1	1	1	选择满刻度校准寄存器	24 位

R/ $\overline{\text{W}}$: 读/写信号。R/ $\overline{\text{W}}$ = 1 是“读”; R/ $\overline{\text{W}}$ = 0 是“写”。

CH2~CH0: 通道选择。这 3 个位用作转换或访问校准系数的通道选择。通道选择见表 3-17。

表 3-17 通道选择

CH2	CH1	CH0	IN+(模拟正)	IN-(模拟负)	类型	校准寄存器对
0	0	0	IN1	IN6	准差分	寄存器对 0
0	0	1	IN2	IN6	准差分	寄存器对 1
0	1	0	IN3	IN6	准差分	寄存器对 2
0	1	1	IN4	IN6	准差分	寄存器对 2
1	0	0	IN1	IN2	全差分	寄存器对 0
1	0	1	IN3	IN4	全差分	寄存器对 1
1	1	0	IN5	IN6	全差分	寄存器对 2
1	1	1	IN6	IN6	测试模式	寄存器对 2

2) 模式寄存器(RS2~RS0=001)

模式寄存器是 8 位寄存器, 可从中读出数据或把数据写入其中。其格式如表 3-18 所示。

表 3-18 模式寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
MD2	MD1	MD0	G2	G1	G0	B0	FSYNC

其中:

MD2~MD0: 工作模式选择位, 见表 3-19。

表 3-19 MD2~MD0 各位的含义

MD2	MD1	MD0	含 义
0	0	0	器件的正常工作模式, 在此模式下器件实现正常转换。这是上电的缺省状态
0	0	1	自校准, 它激活 CH2~CH0 所选择通道的自校准。完成时, 引脚 $\overline{\text{DRDY}}$ 回到“0”
0	1	0	零刻度系统校准。依据在模拟输入端所提供的输入电压, 以所选择的增益实现校准
0	1	1	满刻度系统校准。依据在模拟输入端所提供的输入电压, 以所选择的增益实现校准
1	0	0	CH2~CH0 所选择通道的系统失调校准。完成时, 引脚 $\overline{\text{DRDY}}$ 回到“0”
1	0	1	CH2~CH0 所选择通道的背景校准。完成时, 引脚 $\overline{\text{DRDY}}$ 回到“0”
1	1	0	CH2~CH0 所选择通道的零刻度自校准。完成时, 引脚 $\overline{\text{DRDY}}$ 回到“0”
1	1	1	CH2~CH0 所选择通道的满刻度自校准。完成时, 引脚 $\overline{\text{DRDY}}$ 回到“0”

G2~G0: 增益设置。当 G2G1G0 为 000~111 时, 增益分别为 1~128, 共 8 挡。

B0: 烧断电流。此位为 0 将断开片内的烧断电流, 这是该位的缺省(上电或复位)状态。此位为 1 将激活烧断电流。当它有效时, 烧断电流连接至所选择的模拟输入对, 一个连接到 IN+输入端, 一个连接到 IN-输入端。

FSYNC: 滤波器标志。当该位为“1”时, 滤波器控制逻辑以及校准控制逻辑均保持在复位状态, 模拟调制器也保持在其复位状态。当该位为“0”时, 调制器和滤波器开始处理数据。

3) 滤波器寄存器

AD7714 有两个 8 位滤波器寄存器, 可以从中读出数据或把数据写入其中。滤波器寄存器分 A 型、Y 型和所有型, 其格式见表 3-20。寄存器各位的含义见表 3-21。

表 3-20 滤波器寄存器的格式

RS2、RS1、RS0=010								
B/U	WL	BST	ZERO	FS11	FS10	FS9	FS8	A 型
B/U	WL	BST	CLKDIS	FS11	FS10	FS9	FS8	Y 型
RS2、RS1、RS0=011								
FS7	FS6	FS5	FS4	FS3	FS2	FS1	FS0	所有型

表 3-21 滤波器寄存器各位的含义

位名	各位 的意义
B/U	双极性/单极性选择。此位为 0, 选择双极性工作(默认); 此位为 1, 选择单极性工作
WL	字长度。此位为 0 则当从数据寄存器中读出时选择 16 位字长(缺省), 否则为 24 位字长
BST	电流提升。此位为 0 将减少模拟前端所取的电流, 此位为 1 从 Vcc 提升电流
ZERO	为了确保 A 型器件正常工作, 必须把 0 写入此位
CLKDIS	主时钟禁止位。此位为逻辑 1 将禁止主时钟出现在 CLK _{IO} 引脚
FS11~FS0	滤波器选择。编程这 12 位数据决定滤波器截止频率, 其范围为 19~4000

4) 测试寄存器(RS2~RS0=100)

测试寄存器在测试器件时使用。建议用户不要把此寄存器的任何位从全 0 的缺省(上电或复位)状态加以改变, 因为这将把器件置于其测试模式之一且不能正常工作。如果器件进入其测试模式之一, 那么执行 RESET 将使器件退出此模式。使器件退出其测试模式之一的另一种方法是把 32 个连续的 1 写入器件, 然后把全 0 写入测试寄存器来复位接口。

5) 数据寄存器(RS2~RS0=101)

器件的数据寄存器是只读寄存器, 它包含 AD7714 最近的转换结果。寄存器可编为 16 位或 24 位字长, 这取决于模式寄存器 WL 位的状态。

6) 零刻度校准寄存器(RS2~RS0=110, 上电为 1F4000H)

AD7714 包括 3 个零刻度校准寄存器, 它们被称为零刻度校准寄存器 0、零刻度校准寄存器 1 和零刻度校准寄存器 2。3 个寄存器相互完全独立, 所以在全差分模式下, 对于每一个输入通道有一个零刻度校准寄存器。这些寄存器均为 24 位读/写寄存器, 当写这些寄存器时, 必须写入 24 位; 否则将没有数据传送到寄存器。寄存器与有关的满刻度校准寄存器一起使用以组成寄存器对。

当把器件设置为允许在数字接口访问这些寄存器时, 器件本身不再访问寄存器的系数来校准输出数据。其结果存在这样的可能性: 在访问校准寄存器(读或写操作)之后, 第一个从器件读得的输出数据包含不正确的数据。另外, 当校准正在进行时, 不要企图对校准寄存器进行读或写操作。通过在校准寄存器操作之前使 SYND 输入为低电平或模式寄存器的 FSYNC 位为高电平且在操作完成后分别使它们为高电平或低电平, 可以避免这些影响。

7) 满刻度校准寄存器(RS2~RS0=111, 上电为 5761ABH)

AD7714 包括 3 个满刻度校准寄存器, 它们被称为满刻度校准寄存器 0、满刻度校准寄存器 1 和满刻度校准寄存器 2。这 3 个寄存器互相独立, 所以在全差分模式下, 对于每一个输入通道有一个满刻度校准寄存器。这些寄存器均为 24 位读/写寄存器, 当写这些寄存器时, 必须写入 24 位; 否则将没有数据传送到寄存器。寄存器与有关的零刻度校准寄存器一起使用以组成寄存器对。

当把器件设置为允许在数字接口访问这些寄存器时, 器件本身不再访问寄存器的系数来校准输出数据。

4. 校准过程及滤波

1) 校准过程

如前所述, AD7714 包含许多校准选项。表 3-22 概述了校准类型、包含的操作以及操

作的持续时间。决定校准的结束有两种方法。第1种方法是监视 $\overline{\text{DRDY}}$ 引脚，它在转换时序结束时返回低电平。 $\overline{\text{DRDY}}$ 不但指示转换时序何时完成，而且指示在其数据寄存器中已有有效的新数据。决定校准何时完成的第2种方法是监视模式寄存器的 MD2、MD1 和 MD0 位。当这些位在校准命令之后返回到 0、0、0 时，指示校准时序已完成，它比 $\overline{\text{DRDY}}$ 给出的时间要早。模式位(MD2、MD1 和 MD0)返回到 0、0、0 的时间代表校准的持续时间。

表 3-22 校准操作

校准类型	MD2~MD0	校准序列	模式位持续时间	$\overline{\text{DRDY}}$ 持续时间
自校准	0, 0, 1	内部零刻度校准和满刻度校准	6x1/输出速率	9x1/输出速率
零刻度系统	0, 1, 0	IN 零刻度校准	3x1/输出速率	4x1/输出速率
满刻度系统	0, 1, 1	在所选增益下 IN 满刻度校准	3x1/输出速率	4x1/输出速率
系统失调	1, 0, 0	IN 零刻度校准和满刻度校准	6x1/输出速率	9x1/输出速率
背景校准	1, 0, 1	内部零刻度校准和正常转换	位不复位	6x1/输出速率
零刻度自校	1, 1, 0	零刻度校准	3x1/输出速率	6x1/输出速率
满刻度自校	1, 1, 1	内部满刻度校准	3x1/输出速率	6x1/输出速率

(1) 自校准。通过把适当的数值(0, 0, 1)写入模式寄存器的 MD2、MD1 和 MD0 位，将开始 AD7714 的自校准。在单极性输入范围的自校准模式下，用于决定校准系数的零刻度点采用器件内部短路的差分对的输入(即 $(\text{IN}+) = (\text{IN}-) =$ 内部偏置电压)。对于这种零刻度校准转换，可设置 PGA 以得到所选择的增益(按照模式寄存器中的 G2、G1、G0 位)来实现满刻度校准转换。

(2) 系统校准。系统校准使 AD7714 能补偿系统增益和失调误差，以及它自己的内部误差。完全系统校准过程有两步：零刻度(ZS)系统校准，然后是满刻度(FS)系统校准。

对于完全的系统校准，首先必须把零刻度点提供给转换器。它必须在校准步骤开始之前加至转换器，且在校准步骤完成之前一直保持稳定。一旦在模拟输入端已设置了系统零刻度，那么便可通过把合适的值(0, 1, 0)写入模式寄存器的 MD2、MD1 和 MD0 位来开始零刻度(ZS)系统校准。所选择的增益用来实现零刻度系统标准。

在完全的系统校准时序完成后，可以由它本身实现另外的失调或增益校准，从而调整系统的零基准点或系统增益。校准参数中的一个，或是系统失调，或是系统增益，将不会影响另一个参数。除非器件包含有效的零刻度系统，否则不应当进行满刻度校准。

当在非缓冲模式下使用器件时，系统校准也可用于消除来自模拟输入端源阻抗的任何误差。前端的简单 RC 抗干扰滤波器可能在模拟输入电压上引入误差，但系统校准可用于消除这种误差。

(3) 系统失调校准。系统失调校准随系统校准和自校准两者的变化而变化。在此情况下，零刻度点用与 ZS(零刻度)系统校准完全相同的方法来决定。

在单极性模式下，系统失调校准在转换函数的两个端点之间完成；在双极性模式下，它在中刻度和正满刻度之间完成。

(4) 背景校准。AD7714 也提供背景校准模式。在该模式中，器件把校准步骤与其正常转换时序相交错。在背景校准模式中，器件提供连续的零刻度自校准，而不提供任何满刻

度校准。在此模式中用于决定校准系数的零刻度点与 ZS(零刻度)自校准中的点完全相同。通过把 1、0、1 写入模式寄存器的 MD2、MD1、MD0 位可以调用背景校准模式。当背景校准模式被调用时,它在每次输出更新之后执行零刻度自校准。其优点是器件连续执行失调校准并自动更新其零刻度校准系数。其结果是温度漂移、电源灵敏度和时间漂移对零刻度误差的影响被自动消除。当打开背景校准模式时,器件将保持在此模式,直到模式寄存器的 MD2、MD1、MD0 被改变为止。由于背景校准不执行满刻度校准,因此在把器件置于背景校准模式之前应当完成自校准。在此模式下失调漂移被消除,这使增益漂移成为器件中唯一未消除的误差源。AD7714 随温度变化而产生的增益漂移的典型值为 $0.2 \times 10^{-6}/^{\circ}\text{C}$ 。当器件处于背景校准模式时,不应使用 SYNC 输入端或 FSYNC 位。

(5) 上电和校准。上电时,AD7714 执行内部复位,它把内部寄存器的内容置为已知的状态。上电或复位之后,缺省值装入所有的寄存器。对于校准寄存器,缺省值包括额定的校准系数。但是,为了确保正确校准器件,校准子程序应当在上电之后执行。

2) 滤波

AD7714 包含片内的低通数字滤波器,用来处理器件的和-差调制器的输出。因此,器件不仅有模/数转换功能,而且也提供滤波的功能。在该器件中,模拟滤波与数字滤波是不同的。

首先,由于数字滤波发生在 A/D 转换过程之后,因此它可以消除转换过程期间内部注入的噪声。模拟滤波不能做到这一点。另外,数字滤波器远比模拟滤波器容易编程。根据数字滤波器的设计,可使用户具有对截止频率和输出更新速率编程的能力。

另一方面,模拟滤波器可以消除在模拟信号到达 A/D 之前附加在模拟信号上的噪声。数字滤波器不能做到这一点,即使信号的平均值在极限范围之内,但是加在接近满刻度的信号之上的噪声尖峰有可能使模拟调制器饱和。为了缓和这个问题,AD7714 在和-差调制器和数字滤波器内设置了超范围的顶区,它允许在模拟输入范围之上 5% 的超范围偏差。如果噪声信号大于此值,则必须考虑模拟输入滤波或者减少输入通道电压以便使其满刻度为模拟输入通道满刻度的一半。这将以把动态范围减少 1 位(50%)为代价,提供大于 100% 的超范围能力。

此外,数字滤波器不提供对数字滤波器采样频率整数倍的任何抑制。但是,器件的输入采样对数字滤波器采样频率的倍数提供衰减,因此未衰减的频带实际上发生在输入采样频率 f_s 倍数的周围。于是,未衰减频带发生在 $n \times f_s$ (其中 $n = 1, 2, 3, \dots$)。在这些频率的任何一侧有 $\pm f_3$ dB 宽度的频带(f_3 dB 是数字滤波器的截止频率),噪声无衰减地通过这些频带到达输出端。

(1) 数字后滤波。在 2.2576 MHz 的 f_{CLK} 频率下,片内调制器以 19.2 kHz 的速率输出。此输出速率与已编程的滤波器输出速率相对应。因为输出数据速率高于 Nyquist 准则,所以给定带宽的输出速率将满足大多数应用的需求。但是,可能有某些应用要求较高的数据速率以实现给定的带宽和噪声性能。需要这种较高数据速率的应用将要求在数字滤波器之后跟随后滤波。

例如,如果所需的带宽为 7.86 Hz,但所需的更新速率为 100 Hz,那么可以用 100 Hz 的速率从 AD7714 取得数据,给出 26.2 Hz 的 -3 dB 带宽。后滤波可以用于这种情况以便把带宽和输出噪声减少到 7.86 Hz 带宽的水平,同时保持 100 Hz 的输出速率。

对于低于 1.26 Hz 的带宽,后滤波也可用于减少来自器件的输出噪声。在增益为 128 且带宽为 1.26 Hz 的情况下,输出均方根噪声为 140 nV。这主要是器件噪声或白噪声,而且由于输入被斩波,噪声具有基本上平坦的频率响应。通过把带宽减至 1.26 Hz 以下,可以减少合成通带中的噪声。带宽减小一半可以使输出均方根噪声约减少为原来的 1/1.25。这种附加的滤波将使稳定时间变长。

(2) 模拟滤波。如前所述,数字滤波器不提供对输入采样频率整倍数的任何抑制。但是,由于 AD7714 的过高采样系数,这些频带仅占频谱的一小部分,大部分宽带噪声被滤除。这意味着相对于常规无片内滤波的转换器而言,在 AD7714 前端对模拟滤波的需求大为减少。此外,由于器件 100 dB 的共模抑制性能扩展至数千赫兹,因此在此范围内的共模噪声将有实质性的减少。

如果在 AD7714 前端放置无源元件,那么在非缓冲模式下必须注意确保源阻抗足够低以致在系统中不会引起增益误差。当它在非缓冲模式下使用时,这主要限制了在 AD7714 前端提供的无源抗干扰滤波的数量。但是,当器件在缓冲模式下使用时,大的源阻抗只是简单地引起小的直流失调误差(10 k Ω 的源电阻将产生小于 10 μ V 的失调误差)。因此,如果 AD7714 前端提供的无源模拟滤波使系统需要任何较大的源阻抗,那么建议使器件工作在缓冲模式。

5. 应用说明

1) 系统同步

SYNC输入(或 FSYNC 位)允许用户复位调制器和数字滤波器而不影响任何设置的状态。这允许用户从已知的时间点(即 SYNC 的上升沿或把 1 写入 FSYNC 位的时刻)开始采集模拟输入的信号。

SYNC输入端也可用于实现两个其他功能。如果多个 AD7714 用公共的时钟工作,那么它们可以被同步以便同时更新它们的输出寄存器。输入的下降沿(或写入模式寄存器 FSYNC 位的 1)复位数字滤波器和模拟调制器,并把 AD7714 设为一致的已知状态。在 SYNC 输入为低电平(或 FSYNC 为高电平)时,AD7714 将保持在此状态。在 SYNC 的上升沿(或当把 0 写入 FSYNC 位时),调制器和滤波器离开其复位状态且在下一个时钟边沿器件再次开始采集输入信号。在使用多个 AD7714 的系统中,加至其 SYNC 输入端的公共信号将同步它们的操作。这通常在每一个 AD7714 已完成它自己的校准或已把校准系数装入其中之后才进行。然后输出的更新被同步,各个 AD7714 的输出更新之间最大可能的差为一个主时钟周期。

SYNC输入也可用于启动转换命令,它使 AD7714 工作在常规转换器方式下。在此模式下,SYNC 的上升沿开始转换,DRDY 的下降沿指示何时完成转换。这种方案的缺点是对每一次数据寄存器的更新,必须考虑滤波器的稳定时间。这意味着在此模式下数据寄存器的更新速率将为原来的 1/3。

2) 复位输入

AD7714 的 RESET 输入复位所有的逻辑、数字滤波器和模拟调制器,同时把所有的片内寄存器复位到其缺省状态。当 RESET 输入为低电平时,DRDY 被驱动至高电平,AD7714 忽略至任何一个寄存器的所有通信。当 RESET 输入返回高电平时,AD7714 开始处理数据,

经历 $3 \times 1/\text{输出速率}$ 之后, $\overline{\text{DRDY}}$ 返回低电平, 指示在数据寄存器中有效的新字。但是, 在 $\overline{\text{RESET}}$ 之后, AD7714 用缺省设置状态工作, 通常有必要在 $\overline{\text{RESET}}$ 命令之后设置所有的寄存器并进行校准。

即使在 $\overline{\text{RESET}}$ 输入为低电平时, AD7714 的片内振荡器电路仍继续工作, 在 CLK_O 引脚上主时钟信号可以继续使用。因此, 在用 AD7714 时钟提供系统时钟的应用中, 在 $\overline{\text{RESET}}$ 命令期间, AD7714 仍可产生不间断的主时钟。

3) 待机模式

AD7714 的 $\overline{\text{STBY}}$ 输入端允许用户在不需要提供转换结果时, 把器件置于待机模式。在待机模式下, AD7714 保持其全部片内寄存器(包括数据寄存器)的内容, 数字接口被复位, $\overline{\text{DRDY}}$ 复位至逻辑 1。在该模式下, 不能从器件访问数据。当器件从待机中脱离时, 它开始处理数据并且以 $\overline{\text{STBY}}$ 输入变为高电平开始, 在 $3 \times 1/\text{输出速率}$ 之后, 在数据寄存器中有新的字可供使用。

当器件用外部主时钟工作时, 假如此主时钟被停止, 那么把器件置于待机模式将使总电流的典型值减至 $5 \mu\text{A}$ 。如果在待机模式下外部时钟继续运行, 那么当电源电压为 5 V 时待机电流增加至 $150 \mu\text{A}$ (典型值), 当电源电压为 3.3 V 时增加至 $75 \mu\text{A}$ 。如果把晶体或陶瓷谐振器用作时钟源, 那么当电源电压为 5 V 时, 待机模式下总电流的典型值为 $400 \mu\text{A}$, 电源电压为 3.3 V 时, 其典型值为 $90 \mu\text{A}$ 。这是因为当器件处于待机模式时片内振荡器电路继续运行的缘故。在由 AD7714 时钟提供系统时钟的应用中, 这点是重要的, 即使在其待机模式下, AD7714 仍产生不间断的主时钟。

4) 有关漂移的考虑

AD7714 使用斩波器稳定技术使输入失调漂移为最小。模拟开关中的电荷注入和采样结点处的直流泄漏电流是转换器中失调电压漂移的主要来源。直流输入泄漏电流基本上与所选择的增益无关。转换器中增益漂移主要取决于内部电容器的温度漂移, 它不受泄漏电流的影响。

任何时候通过重新校准转换器或使器件工作在背景校准模式, 可以消除由于失调漂移或增益漂移而产生的测量误差。使用系统校准模式可以使信号调理电路中失调和增益误差为最小。积分和差分线性误差不受温度变化的影响。

5) 数字接口

AD7714 的串行接口包含 5 个信号: $\overline{\text{CS}}$ 、 SCLK 、 DIN 、 DOUT 和 $\overline{\text{DRDY}}$ 。 DIN 线用于把数据传送到片内寄存器。 DOUT 线用于输出片内寄存器的数据。 SCLK 是器件的串行时钟输入, 所有的数据传送(在 DIN 或 DOUT)相对于此 SCLK 信号而发生。 $\overline{\text{DRDY}}$ 线用作状态信号, 它指示何时数据从 AD7714 数据寄存器中读出。当输出寄存器中有新的数据字可供使用时, $\overline{\text{DRDY}}$ 变为低电平。当对数据寄存器的读操作完成时, 它复位至高电平。在输出寄存器更新之前它也变为高电平以便指示尚未从器件中读出数据, 从而确保当寄存器正在更新时不会企图读出数据。 $\overline{\text{CS}}$ 用于选择器件。在许多器件连接到串行总线的系统中, $\overline{\text{CS}}$ 也可用于对 AD7714 译码。

通过把 $\overline{\text{CS}}$ 输入端连接到低电平, AD7714 串行接口可工作于 3 线模式。在此情况下, SCLK 、 DIN 和 DOUT 线用于与 AD7714 通信, 通过查询通信寄存器的 MSB 可以获得 $\overline{\text{DRDY}}$ 的状态。

图 3-17 表示 $\overline{\text{CS}}$ 用于器件的译码时, 与 AD7714 接口的时序图。图 3-17(a)适用于对 AD7714 输出移位寄存器的读操作; 图 3-17(b)表示对输入移位寄存器的写操作。这两个图均适用于 POL 输入端为逻辑高电平的情况。当工作于 POL 输入端为逻辑低电平时, 只需简单地把图中所示的 SCLK 波形反相即可。即使在第 1 次读操作之后, $\overline{\text{DRDY}}$ 线返回高电平, 仍有可能在输出寄存器中两次读出同样的数据。但是, 应当注意, 在下次输出更新将要发生之前, 要确保完成读操作。

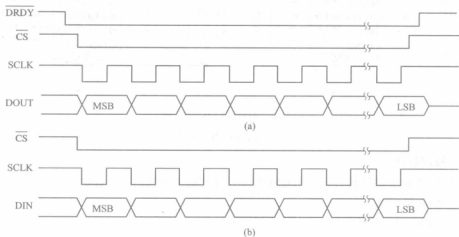


图 3-17 AD7714 的读/写时序(POL = 1)

(a) 读时序; (b) 写时序

通过使用器件的 $\overline{\text{RESET}}$ 输入可以复位串行接口。它也可以通过在 DIN 输入端上写一系列 1 来实现。如果对 DIN 线写逻辑 1 至少达 32 个串行时钟周期, 那么串行接口将复位。

3.4.2 应用电路与编程

1. 温度测量

AD7714 的一个应用领域是温度测量。图 3-18(a)表示热电偶与 AD7714 的连接。在此应用中, AD7714 工作在其缓冲模式下以便允许在前端连接大的去耦电容, 从而可消除在热电偶引线上的噪声。当 AD7714 工作在缓冲模式时, 它具有较小的共模范围。为了使热电偶的差分电压置于合适的共模电压之上, 把 AD7714 的 IN2 输入偏置到基准电压+2.5 V。

2. 数据采集

AD7714 的 3 个差分通道(或 5 个准差分通道)适用于低带宽、高分辨率数据采集系统。此外, 3 线数据接口使此数据采集系统的前端能够只用 3 个光隔离器便实现隔离。假如 AD7714 模拟输入端的输入信号全部为正极性, 那么整个系统可以用单个+3 V 或+5 V 电源工作。AD7714 低功耗运用可确保隔离元件两端的功率非常小。图 3-18(b)表示隔离数据采集系统中 AD7714 的用法。

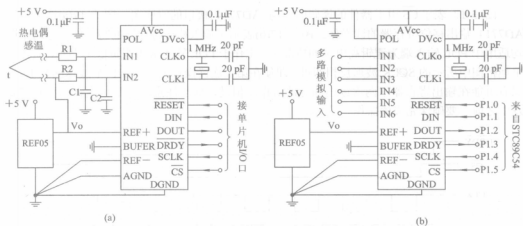


图 3-18 AD7714 的应用实例

(a) 测温电路; (b) 数据采集电路

3. 程序设计

按照图 3-18(a)的连接方式, 用 C51 程序设计规则实现的相关函数如下:

```
#include <reg52.h>

#define SAMPLES 1000 /*采样延时常数*/

sbit RESET = P1^0 ; /*定义复位引脚*/
sbit DIN = P1^1 ; /*定义输入引脚*/
sbit DOUT = P1^2 ; /*定义输出引脚*/
sbit CS = P1^5 ; /*定义片选引脚*/
sbit SCLK = P1^4 ; /*定义输出时钟*/
sbit DRDY = P1^3 ; /*定义 DRDY 引脚*/

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

/*功能: 向 AD7714 写入一个字节。参数: byte 为写入的数据*/
void WriteAD7714(unsigned char byte)
{
    unsigned char dat, i; /*分别用于传递形参*/
    dat=byte; /*发送数据*/
    CS=1; SCLK=1; /*初始化*/
    CS=0; /*CS=0, 准备开始送数*/
    for(i=0; i<8; i++)
    {
```

```

        if(dat&0x80) DIN=1;
        else DIN=0;
        delay(4); SCLK=1; /*时钟线=1*/
        delay(4); SCLK=0; /*时钟线=0*/
        delay(4); dat<<=1; /*左移 1 位*/
    }
    CS=1; /*CS=1, 结束传送数据*/
}

/*功能: 读 AD7714, 返回值是长整型数据。形参: leth 是数据长度*/
unsigned long Read_AD7714_leth(unsigned char leth)
{
    unsigned long x=0;
    unsigned char j;
    CS=1; SCLK=1; DRDY=0; /*初始化*/
    while (DRDY==1); /*DRDY=1, 等待*/
    CS=0; /*CS=0, 准备读数据*/
    for(j=0; j<leth; j++)
    {
        x<<=1; delay(4);
        SCLK=1; delay(4);
        if(DIN==0) x |= 0xffffffe;
        else x |= 0x00000001;
        SCLK=0; /*时钟线置低*/
    }
    CS=1; /*CS=1, 读数据结束*/
    return(x); /*返回 A/D 的值(leth 长的数据)*/
}

void main (void) /*主函数*/
{
    unsigned int i;
    unsigned long ad_data;
    RESET=0; /*复位*/
    delay(0x90);
    RESET=1;
    CS=1; /*CS=1(高电平)*/
    WriteAD7714(0x27); /*设置为测试模式(IN6+,IN6-), 下一操作为滤波器高 8 位*/
    WriteAD7714(0xA0); /*设置为双极性, 24 bit*/
    WriteAD7714(0x37); /*设置下一操作为滤波器低 8 位*/
    WriteAD7714(0x40); /*设置滤波器低 8 位*/
    WriteAD7714(0x17); /*设置下一操作为模式寄存器*/
}

```

```

WriteAD7714(0x20);          /*设置: 增益为 1, 自校准模式*/
while(DRDY==1);             /*等待 DRDY=0*/
for(i=0; i<SAMPLES; i++);   /*等待采样*/
while(1)
{
    WriteAD7714(0x5c);        /*设置(IN1+,IN2-), 下一操作是读数据寄存器的值*/
    ad_data=Read_AD7714_lenth(16); /*读 lenth 位的 A/D 值*/
    delay(0x50);
    printf("24 bitDATA=%d\n", ad_data);
}
}

```

3.5 AD5320 微功耗满幅 12 位串行 D/A 转换器

3.5.1 硬件与功能描述

AD5320 是单通道 12 位电压输出型 D/A 转换器, 片内具有精密的放大器使输出可达满幅电压输出。AD5320 采用灵活的 3 线串行接口, 时钟速率可达 30 MHz。AD5320 的基准由电源输入端产生, 可提供很宽的动态输出范围。器件内包含一个上电复位电路, 可保证 DAC 输出上电至 0 V 直到器件发生有效写操作。器件还包含掉电特性, 在 +5 V 时可将电流消耗的典型值降至 200 nA 以下, 并在掉电状态中提供软件可选的输出负载。AD5320 还具有施密特触发的低功耗输入电路。

AD5320 可广泛用于便携式电池供电设备、数字增益和失调调整电路、可编程电压和电流源及数字衰减器等领域。

1. 主要性能特点

- (1) 转换精度为 12 位 D/A 输出。
- (2) 输出建立时间的典型值为 12 μ s。
- (3) 转换速率的典型值为 1 V/ μ s。
- (4) 采用 3 线串口与标准的 SPITM、QSPITM 和 DSP 接口兼容。
- (5) 宽电源范围为 +2.7~+5.5 V。
- (6) 在 +5 V 时, 耗电典型值为 140 μ A; 在 +3 V 时, 耗电典型值 115 μ A。
- (7) 在 +5 V 时的功耗从 0.7 mW 减至 1 μ W。
- (8) 工作温度为 -40~+105 $^{\circ}$ C。

2. 内部结构与引脚功能

AD5320 采用 CMOS 工艺制造, 其结构包含 DAC 寄存器、电阻网络、输入电路、输出电路和控制电路等, 如图 3-19(a)所示。电源(Vcc)用作基准输入, 理想的输出电压为

$$V_O = V_{CC} \times \frac{D}{4096}$$

其中, D 为装载到 DAC 寄存器的二进制代码的十进制形式, 其范围是 0~4096。

输出缓冲放大器可在其输出产生范围为 $0 \sim V_{CC}$ 的满电源幅度电压输出, 也能驱动 $2 \text{ k}\Omega$ 负载。

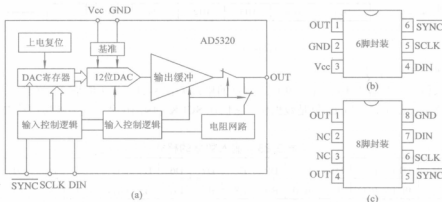


图 3-19 内部结构与引脚排列

(a) 内部结构; (b) 6 脚排列; (c) 8 脚排列

AD5320 具有 6 脚 SOT-23 封装和 8 脚微型 SOIC 封装, 其引脚排列见图 3-19(b)与(c)。其中:

- (1) OUT 是 DAC 的模拟输出电压。
- (2) GND 是器件中所有信号的地参考点。
- (3) Vcc 是电源输入端。
- (4) DIN 是串行数字输入, 数据在串行时钟输入的下降沿随时钟移入 16 位寄存器。
- (5) SCLK 是串行时钟输入, 数据的传送速率高达 30 MHz。
- (6) SYNC 是输入数据的帧同步信号, 电平触发控制输入(低电平有效)。当 SYNC 变为低电平时, 它会使得输入移位寄存器, 数据在后续时钟的下降沿被移入。DAC 在第 16 个时钟周期后被更新, 除非在此时钟边沿之前 SYNC 变为高电平。在这种情况下, SYNC 的下降沿用作一次中断, 而写信号则被 DAC 忽略。

3. 使用说明

AD5320 具有 3 线串行接口(SYNC、SCLK 和 DIN), 与 SPI、QSPI 及大多数 DSP 兼容。图 3-20 为典型写操作时序。将 SYNC 拉至低电平可启动写序列。来自 DIN 线的数据在 SCLK 的下降沿随时钟送入 16 位移位寄存器。时钟频率可高达 30 MHz, 使 AD5320 能与高速 DSP 兼容。在第 16 个时钟下降沿, 最后一个数据输入将改变 DAC 寄存器的内容(或改变工作方式)。在这个阶段, SYNC 线可保持低电平或被拉为高电平。在两种情况下, SYNC 必须在下一个写序列前被拉为高电平至少 33 ns, 以使它的下降沿可启动又一个写序列。为了使器件工作时功耗更低, 应使 SYNC 在写序列之间置为低电平。在下次写序列之前, 必须使 SYNC 返回为高电平。

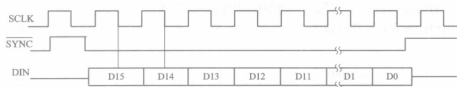


图 3-20 串行写操作

1) 输入移位寄存器

输入移位寄存器为 16 位宽。其输入数据的格式见表 3-23。前 2 位是“无关位”，接下来的 2 位是控制位，最后 12 位是数据位，它们在 SCLK 的第 16 个下降沿被传送给 DAC 寄存器。

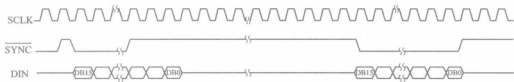
表 3-23 输入数据的格式

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
x	x	PD1	PD0	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

表 3-23 中，PD1、PD0 是工作方式选择位，为 00 时是正常方式，为 01 和 10 时是掉电方式，为 11 时是三态。

2) SYNC 中断

在正常的写序列中， $\overline{\text{SYNC}}$ 线至少要保持低电平一直到 SCLK 的第 16 个下降沿，DAC 在这第 16 个下降沿被更新。但是，如果 SYNC 在第 16 个下降沿之前被拉为高电平，则意味着写序列的中断。此时移位寄存器复位，写序列视做无效，DAC 寄存器内容的更新或工作方式的改变均不发生，如图 3-21 所示。

图 3-21 $\overline{\text{SYNC}}$ 中断

3) 上电复位

AD5320 包含一个上电复位电路，可在上电期间控制输出电压。DAC 寄存器填充 0，输出电压为 0 V，且一直保持到对 DAC 发生有效的写序列。这一点对于在上电过程中需要知道 DAC 输出状态的应用领域非常重要。

4) 掉电方式

AD5320 具有四种不同的工作方式，这些方式可由软件对控制寄存器中的两位(D13 和 D12)编程而得(见表 3-23 中的 PD1、PD0 位)。当两位都置为 0 时，器件正常工作，其正常功耗在 +5 V 时为 140 μA 。但是，对于三种掉电方式，电源电流在 +5 V 时降至 200 nA (+3 V 时为 50 nA)。不仅如此，输出级在内部也从放大器输出切换到阻值已知的电阻网络。其优点是在器件处于掉电方式时输出阻抗是已知的，即该位为“01”时，输出内阻是 1 k Ω ，该位为“10”时，输出内阻是 100 k Ω ，该位为“11”时，输出是开路(三态)状态。

当激活掉电方式时,偏置电路、输出放大器、电阻网络和其他有关的线性电路都被关闭,而 DAC 寄存器的内容却不受影响。退出掉电方式所需的时间在 $V_{CC}=5\text{ V}$ 时和 $V_{CC}=3\text{ V}$ 时的典型值分别为 $2.5\text{ }\mu\text{s}$ 和 $5\text{ }\mu\text{s}$ 。

5) 电源旁路和接地

在印制电路板上适当考虑电源和地回路的布局对于提高电路的精度很有好处。在包含 AD5320 的实际电路板中,数字和模拟电路部分应该分开,各自有自己的电路板空间。如果系统中其他器件要求模拟地(AGND)和数字地(DGND)相连,那么应该将它们只连接在一点,该接地点应该尽可能靠近 AD5320。AD5320 的电源应该用 $10\text{ }\mu\text{F}$ 和 $0.1\text{ }\mu\text{F}$ 的电容器旁路。电容应尽可能靠近器件,最理想的是将 $0.1\text{ }\mu\text{F}$ 的电容放置在 AD5320 的正上方。 $10\text{ }\mu\text{F}$ 电容为钽电容。 $0.1\text{ }\mu\text{F}$ 电容最好具有较低的等效串联电阻和等效串联电感,例如普通的陶瓷电容。对于内部逻辑切换引起的瞬变电流而产生的高频, $0.1\text{ }\mu\text{F}$ 的电容可提供一条接地的低阻抗路径。电源线本身应该具有尽可能宽的布线以提供低阻抗回路并降低电源线上的低频干扰。时钟和其他快速开关数字信号应该用数字地将自己与电路板上其他部件屏蔽开来。如有可能,尽量避免数字和模拟信号之间的串扰。当布线穿过电路板的反面时,保证它们互相成直角以减少对电路板的反馈作用。最佳的电路板布局方式是元件面专用作接地面,而信号线则布置于焊接面上。

3.5.2 应用电路与编程

1. AD5320 与单片机的接口

图 3-22(a)是 AD5320 和 80C51 单片机的串行接口。接口可进行如下配置:80C51 的 P1.0 驱动 AD5320 的 SCLK, P1.1 驱动器件的串行数据线 DIN, SYNC 信号来自于端口 P1.2。

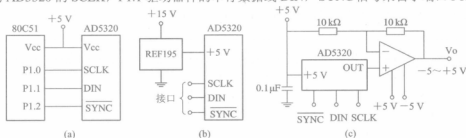


图 3-22 AD5320 的三种典型连接电路

(a) 与 80C51 的连接; (b) 用专用电源供电; (c) 双极性连接

2. 为 AD5320 提供专用电源

图 3-22(b)是将 REF195 用作 AD5320 电源的连接电路。因为 AD5320 要求电源电流非常低,所以可以选择用 REF195 电压基准(对于 5 V 用 REF195,而对于 3 V 则用 REF193)为器件提供所需的电源。如果电源有较大干扰或系统电源电压的值不是 5 V 或 3 V (例如是 15 V),那么这一点就显得尤为重要。REF195 将为 AD5320 输出稳定的电源电压。如果使用 REF195,那么它需要给 AD5320 提供的电流为 $140\text{ }\mu\text{A}$ 。此时 DAC 的输出无负载。当 DAC 输出加载时,REF195 还需要向负载提供电流。

3. AD5320 的双极性工作

AD5320 是为单电源工作而设计的,若采用图 3-22(c)所示的连接方法,则可实现一个双极性模拟输出电压。采用 AD820 或 OP295 作为输出放大器可使其输出达到满电源幅度。

4. AD5320 带有光隔离接口的应用

在工业环境中的过程控制应用领域,非常有必要使用光电器件来保护和隔离控制电路免受任何共模电压的损害。光隔离器提供超过 3 kV 的隔离电压。由于 AD5320 使用了 3 线串行逻辑接口,因此仅需要 3 个光隔离器就能实现隔离,如图 3-23 所示。器件的电源也需要被隔离,用一个变压器即可做到这一点。在变压器的 DAC 一侧,一个 +5 V 稳压器可提供 AD5320 所需的 +5 V 电源。

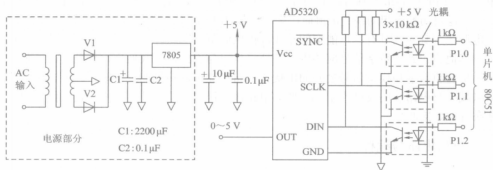


图 3-23 带有光隔离的应用

5. 编程方法

对于图 3-23 的连接,实现一个“锯齿波”的程序如下:

```
#include <reg52.h>

sbit F_SYNC=P1^0;           /*定义同步引脚*/
sbit F_SCLK=P1^1;           /*定义时钟引脚*/
sbit F_DIN =P1^2;           /*定义输入引脚*/

void delay(unsigned char x)  /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

/*功能: 向 AD5320 写入两个字节数据。参数: data_2 写入的数据*/
void WriteAD5320(unsigned int data_2)
{
    unsigned char i;         /*分别用于传递形参*/
    unsigned int dat;
    dat=data_2 & 0x3fff;     /*发送数据*/
    F_SYNC=0; F_SCLK=0;      /*初始化,实际上电路已取反*/
    F_SYNC=1;                /*SYNC=0 准备开始送数*/
```

```

for(i=0; i<16; i++)
{
    if(dat & 0x8000) F_DIN=0;    /*DIN=1*/
    else F_DIN=1;                /*DIN=0*/
    delay(4); F_SCLK=1;          /*SCLK=0*/
    delay(4); F_SCLK=0;          /*SCLK=1*/
    delay(4); dat<<=1;           /*左移1位*/
}
F_SYNC=0;                        /*SYNC=1, 结束传送数据*/
}

void main (void)                  /*这是一个产生锯齿波的测试程序*/
{
    unsigned int da_data;
    F_SYNC=0;                      /*SYNC=1(高电平)*/
    da_data=0;
    while (1)                      /*产生锯齿波*/
    {
        da_data=da_data+8;          /*数据每次加8为步长*/
        if (da_data>=4096) da_data=0;
        da_data = da_data & 0x03ff;
        WriteAD5320(da_data);       /*向 D/A 写入数据, 进行转换*/
    }
}

```

3.6 TLC5618 可编程双路 12 位串行 D/A 转换器

3.6.1 硬件与功能描述

TLC5618 是双路带有缓冲基准输入的 12 位电压输出型 D/A 转换器。两个转换器同时更新, D/A 输出电压范围为基准电压的两倍, 且为单调变化。TLC5618 可在单 5 V 电源下工作, 内部包含上电复位功能以确保可重复启动。通过 CMOS 兼容的 3 线串行总线可对 TLC5618 实现数字控制。数字输入的特点是带有斯密特触发器, 具有高的噪声抑制能力。数字通信协议包括 SPITM、QSPI 和 Microwire 标准。

该器件可广范应用于便携式电测仪表、数字增量调整电路、电池供电电路、模拟控制和移动电话等领域。

1. 主要性能特点

- (1) 典型的建立时间为 12.5 μ s;
- (2) 转换精度为双路 12 位;

- (3) DAC 输出类型为双路电压输出;
- (4) 输入阻抗的典型值为 $10\text{ M}\Omega$;
- (5) 数字接口为 3 线串行接口;
- (6) 数据输入速率的典型值为 1.21 MHz ;
- (7) 电源范围为 $4.5\sim 5.5\text{ V}$;
- (8) 参考电压的典型值为 2.048 V ;
- (9) 电源噪声抑制比的典型值为 65 dB ;
- (10) 工作温度范围为: C 挡, $0\sim 70^{\circ}\text{C}$; I 挡, $-40\sim +85^{\circ}\text{C}$ 。

2. 内部结构与引脚说明

TLC5618 的内部结构与引脚排列如图 3-24 所示。其中, DIN 是串行数据输入; SCLK 是串行时钟输入; $\overline{\text{CS}}$ 是片选端, 低电平有效; OUTA 是 A 通道模拟输出; OUTB 是 B 通道模拟输出; REFIN 是基准电压输入端; Vcc 是电源端, 一般接 $+5\text{ V}$; AGND 是模拟地。

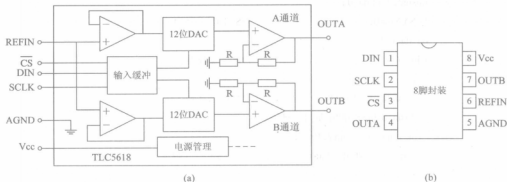


图 3-24 TLC5618 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

3. 工作时序

TLC5618 的工作时序如图 3-25 所示。为了使时钟馈通为最小, 当 $\overline{\text{CS}}$ 为高电平时, 加在 SCLK 端的输入时钟应当禁止为低电平。

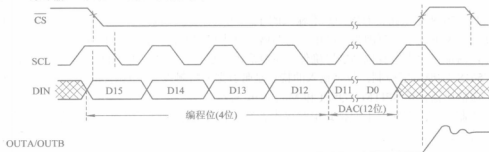


图 3-25 TLC5618 的工作时序

4. 使用说明

TLC5618 通过运放缓冲将 12 位数据转换为模拟电压。TLC5618 的输出极性与基准输入

相同。输出电压表达式为

$$V_o = 2 \times \text{REF} \times \frac{\text{CODE}}{4096}$$

式中, REF 是外加的基准电压; CODE 是输入的 12 位二进制代码。上电时内部电路把 DAC 寄存器复位至 0。

1) 缓冲放大器

输出缓冲放大器在满幅时可达电源电压幅度的输出, 它带有短路保护并能驱动具有 100 pF 负载电容的 2 kΩ 负载。到达最终值±0.5 LSB 以内的建立时间为软件可选的 15 μs 或 3 μs(典型值)。

2) 外加基准

基准电压决定 DAC 满幅输出。外加基准由于经输入电阻约 10 MΩ(电容约 5 pF)的缓冲器隔离, 因此基准与 DAC 输入电阻和代码无关, 从电路上保证了 D/A 转换的精度。

3) 逻辑接口

数据输入适用于大多数标准的高速 CMOS 逻辑接口, 最大串行时钟速率约 1.21 MHz。当 $\overline{\text{CS}}$ 为低电平时, 输入数据由时钟定时以最高有效位在前的方式输入 16 位移位寄存器。SCLK 输入的下降沿把数据移入输入寄存器, 然后 $\overline{\text{CS}}$ 的上升沿把数据送到 DAC 寄存器。所有 CS 的跳变应当发生在 SCLK 输入为低电平时。16 位数据可以表 3-24 所示的格式写入 TLC5618。

表 3-24 TLC5618 数据格式

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
C15	C14	C13	C12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

表 3-24 中, B0~B11 是 12 位被转换的数据; C12~C15 是编程(设置)位, 见表 3-25。

表 3-25 可编程位的功能

可 编 程 位				器 件 功 能
C15	C14	C13	C12	
1	x	x	x	把串行数据写入锁存器 A 并用缓冲器更新锁存器 B
0	x	x	0	写锁存器 B 和双缓冲锁存器
0	x	x	1	仅写双缓冲锁存器
x	1	x	x	15 μs 建立时间
x	0	x	x	3 μs 建立时间
x	x	0	x	上电操作
x	x	1	x	断电方式

表中, x 为任意值(0 或 1)。

对于表 3-25 可编程位 C15~C12 的功能, 实际上有三种数据传送方式。

(1) 写锁存器 A, 更新锁存器 B (C15 = 1, C12 = x)。将串行接口寄存器的数据(SIR)写入锁存器 A 通道, 双缓冲锁存器的内容写入锁存器 B 通道, 双缓冲器的内容不受影响。此控制位状态允许两个 DAC 同时更新。此方式见图 3-26(a)。

(2) 写锁存器 B 和双缓冲器($C15=0$, $C12=0$)。SIR 数据写入锁存器 B 和双缓冲器, 锁存器 A 不受影响。此方式见图 3-26(b)。

(3) 仅写双缓冲器($C15=0$, $C12=1$)。SIR 数据仅写入双缓冲器, 锁存器 A 和 B 的内容不受影响。此方式见图 3-26(c)。

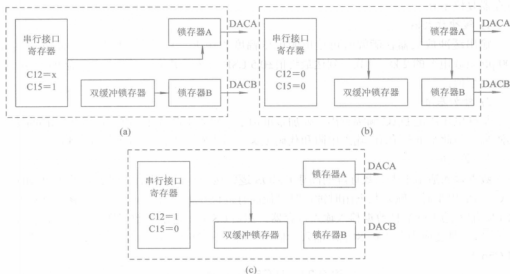


图 3-26 锁存器 A 和 B 的控制方式

(a) 写锁存器 A, 更新锁存器 B; (b) 写锁存器 B 和双缓冲器; (c) 仅写双缓冲器

通常在写操作之后只有一个 DAC 输出可以改变。双缓冲器允许在单次写操作之后两个 DAC 输出都改变。这可通过以下两步来实现:

① 执行仅写双缓冲器以储存新的 DACB 数据, 不改变通道 A 和 B 的输出($C15=0$, $C12=1$)。

② 在上一步骤之后执行写锁存器 A($C15=1$, $C12=x$)。这将把 SIR 的数据写入锁存器 A 并且也把双缓冲器的内容写入锁存器 B。于是两个 DAC 同时接收其新数据, 而且两个 DAC 的输出同时开始改变。

4) 操作举例

(1) 把锁存器 A 的数据以全 0 变为全 1。假设锁存器 A 开始时为全 0(例如在上电之后), 那么通过把(1000 1111 1111 1111)写入 TLC5618 即可。此写操作将不改变双缓冲锁存器的内容以及 DACB 的输出。

(2) 把锁存器 B 的数据从全 0 变为全 1。假设锁存器 B 开始时为全 0(例如在上电之后), 那么通过把(0d00 1111 1111 1111)写入器件即可。此操作将不改变锁存器 A 的内容以及 DACA 的输出。

(3) 两个 DAC 输出的双缓冲改变。假设 DACA 和 DACB 开始时都为 0(例如在上电之后), 如果要把 DACA 变为半满度, 把 B 变为满度而且输出同时开始上升, 那么可按以下步骤操作:

① 把(0001 1111 1111 1111)写入器件(实际上是先存入双缓冲锁存器),这个写操作不影响锁存器 B 的内容以及 DACA 的输出。从快速至慢速方式或从慢速至快速方式的变化将改变电源电流,这将引起输出闪变,因此 C14(在数据字中用 d 表示)应当设置成使速度保持原先写操作所设置的速度。

② 把(1000 1000 0000 0000)写入串行接口。位 C14 可以为零以选择慢速方式或为 1 以选择快速方式。这将把半满度代码(1000 0000 0000)写入锁存器 A 同时把满度代码从双缓冲器复制到锁存器 B。于是在第二个写操作之后两个 DAC 输出开始上升。

3.6.2 应用电路与编程

1. 通用串行接口

TLC5618 的 3 线接口与 SPI、QSPI 以及 Microwire 串行标准兼容。图 3-27 所示是 TLC5618 与 STC89C52 单片机的一种连接。为了远距离传送,该电路将数字信号线用光电器件隔离,这样可减少干扰。

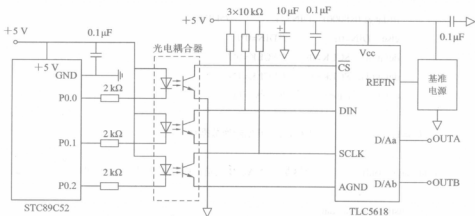


图 3-27 TLC5618 与 STC89C52 单片机的一种连接

通常使用分离的模拟和数字地平面的印制电路板可提供最佳的系统性能。两个地平面应当在低阻抗的电源处连接在一起。把 DAC AGND 端连接到系统模拟地平面,确保模拟地电流流动良好且可以实现最佳的接地连接。另外,应当在 Vcc 与 AGND 之间连接 0.1 μF 陶瓷旁路电容,且应当用短的引线安装在尽可能靠近器件的地方。

当不使用 DAC 时,把 DAC 寄存器设置为全 0 可以使基准电阻阵列和输出负载所消耗的功率最小。

2. 编程方法

对于图 3-27 所示的连接,实现一个 A 与 B 通道同时输出 D/A 变换结果的程序如下:

```
#include "reg52.h"
sbit CS = P0^0; /*片选引脚*/
sbit SCLK = P0^2; /*定义时钟引脚*/
sbit DIN = P0^1; /*定义输入引脚*/
```



```

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

/*功能是向 TLC5618 写入一个整型数据, 参数 data_2 为写入的数据*/
void WriteTLC5618(unsigned int data_2)
{
    unsigned char i;
    unsigned int dat;
    dat=data_2;          /*发送数据*/
    CS=1; SCLK=1;        /*初始化*/
    CS=0;                 /*准备开始送数*/
    for(i=0; i<16; i++)
    {
        if(dat & 0x8000) DIN=1;      /*DIN=1*/
        else DIN=0;                 /*DIN=0*/
        delay(4); SCLK=0;           /*SCLK=0*/
        delay(4); SCLK=1;           /*SCLK=1*/
        delay(4); dat<<=1;          /*左移一位*/
    }
    CS=1;                      /*CS=1, 开始转换数据*/
}

void main (void)              /*这是一个 A、B 通道同时输出代码为 0x555 的电压值的测试程序*/
{
    unsigned int da_data;
    CS=1;
    da_data = 0x555;           /*给双缓冲锁存器写 0x555*/
    WriteTLC5618(da_data); /*目的是要写入 B 锁存器*/
    /*给 A 锁存器写 0x555, 并将双缓冲器的内容写入 B 锁存器*/
    da_data = 0x555;
    WriteTLC5618(da_data); /*A、B 同时输出 0x555 的被转换电压*/
    while(1);
}

```

3.7 AD9764 高精度高速 D/A 转换器

3.7.1 硬件与功能描述

AD9764 是 14 位高性能低功耗的数/模转换器。AD9764 有很好的直流和交流性能, 它

能支持高达 125 MHz 的数据转换速率。AD9764 采用先进的 CMOS 工艺, 将一个分段式电流源结构和特殊的开关技术相结合, 减少了杂波分量, 提高了动态性能; 边沿触发锁存和 1.2 V 带补偿的内部基准提供了完整的单片 DAC 解决方案; 灵活的电源选择支持 +3 V 和 +5 V CMOS 逻辑系列电路; 提供差分电流输出, 可支持单端或差分应用; 两个输出电流匹配以确保在差分输出结构中提高其动态性能; 电流输出可直接与输出电阻相接, 提供两个互补的单端电压输出, 也可直接输入变压器, 输出电压是 1.25 V。

AD9764 可采用片内基准和外部基准源驱动。内部控制放大器提供宽达 10:1 的调整跨度, 并允许 AD9764 的满量程电流在 2~20 mA 间调整, 同时还能保持良好的动态性能。这样 AD9764 可在低功耗下工作, 还可具有在大于 20 dB 的范围调整外部增益的能力。

AD9764 为 28 脚 SOIC 封装, 可在工业温度范围内工作。

1. 主要性能特点

- (1) 转换速率高达 125 MHz;
- (2) 转换精度为 14 位;
- (3) 差分电流输出为 2~20 mA;
- (4) 输出阻抗 > 100 k Ω ;
- (5) 片内基准为 1.2 V;
- (6) 参考输入阻抗的典型值为 1 M Ω ;
- (7) 电源电压为 2.7~5.5 V;
- (8) 功耗 < 160 mW (5 V 供电, 输出电流为 20 mA 时);
- (9) 工作温度为 -40~+85 $^{\circ}$ C。

2. 内部结构与引脚说明

图 3-28(a) 给出了 AD9764 的一个简化了的内部结构框图。AD9764 包含一个大的 PMOS 电流源阵列, 该阵列具有提供 20 mA 总电流的能力。这个阵列被分成 31 个相等的电流, 它们形成了五个最高有效位 (MSB)。接着的四个位即中间位包括 15 个相等的电流源, 它们的值是一个 MSB 电流源的 1/16。其余的 LSB 是中间位电流源的二进制权的一部分。以电流源来实现中间和最低位, 不用 R-2R 阶梯电阻, 对多通道或小幅度信号而言, 优化了它的动态性能, 并且有助于保持 DAC 的高输出阻抗 (> 100 k Ω)。所有这些电流源通过 PMOS 的差分电流开关转换到两个端点的任何一个 (即 IOUTA 或 IOUTB) 上。这些开关基于这样的一种新结构, 明显地减少了各种定时误差。

AD9764 的模拟和数字部分具有独立的电源输入 (即 AVcc 和 DVcc), 它们分别工作在 2.7~5.5 V 的电压范围内。数字部分能够工作在 125 MHz 时钟速率上, 它包括边沿触发锁存和分段译码逻辑电路。模拟部分包括 PMOS 电流源、差分开关、一个 1.2 V 电压基准和一个基准控制放大器。

满量程输出电流由基准控制放大器所调整, 通过一个外部电阻 Rset, 能够在 2~20 mA 间变化, 外部电阻与基准控制放大器和电压基准 Ve 相接。该电阻设定基准电流 Ie, 该基准电流应是分段电流源电流的适当倍数。满量程电流 IOUT 是 Ie 的 32 倍。

AD9764 的引脚排列如图 3-28(b) 所示。其引脚功能见表 3-26。

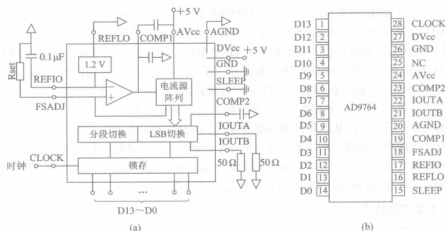


图 3-28 AD9764 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

表 3-26 AD9764 引脚功能

引脚	名称	状态	含 义
1~14	D13~D0	三态	数据总线。其中, D0 为最低位, D13 为最高位(MSB)
15	SLEEP	输入	掉电控制输入, 高电平有效
16	REFLO	输入	使用内部 1.2 V 基准时, 该脚接地。接 AVcc 时, 禁止内部基准
17	REFIO	—	接外部基准输入端。用内部基准时, 该脚接 0.1 μ F 的电容到 AGND
18	FSADJ	输入	满量程电流输出调整
19	COMP1	—	带宽/噪声下降节点。一般应接 0.1 μ F 电容到 AVcc 引脚
20	AGND	—	模拟公共地
21	IOUTB	输出	互补 DAC 电流输出。当所有数据位是 0 时, 为满量程电流
22	IOUTA	输出	DAC 电流输出。当所有数据位是 1 时, 为满量程电流
23	COMP2	—	驱动电路内部偏置节点。应接 0.1 μ F 电容到 AGND 上去耦
24	AVcc	—	模拟电源电压(2.7~5.5 V)
25	NC	—	无内部连接
26	GND	—	数字公共地
27	DVcc	—	数字电源电压(2.7~5.5 V)
28	CLOCK	输入	时钟输入。在时钟的正边沿, 数据被锁存

3. 使用基本描述

1) 电压基准

AD9764 包括一个内部 1.2 V 电压基准, 能够很容易地被外部基准所禁止和取代。REFIO 当成一个输入或输出端, 这取决于选择内部还是外部基准。如果 REFLO 与 AGND 相连, 如图 3-28(a)所示, 则内部基准被激活, REFIO 提供一个 1.2 V 的输出。此时, 内部基准必须用 0.1 μ F 的陶瓷电容或更大容量的电容接于 REFIO 去耦。同样, 如果需要任何附加的电压基准, 则 REFIO 应接外部放大器来缓冲, 该放大器有一个小于 100 nA 的输入电流。

通过把 REFLO 接到 Vcc 上可以用来禁止内部基准。当使用外部基准时,外部基准从 REFIO 端输入。

2) 基准控制放大器

AD9764 同样包括一个内部基准放大器,它用于调整 DAC 的满量程输出电流 I_{out} 。控制放大器构成一个 V-I 转换器,允许 I_{out} 在 2~20 mA 范围内有一个宽(10:1)的调整跨度。 I_{out} 的宽的调整跨度的第一个优点是功耗有直接的关系,功耗与 I_{out} 成正比;第二个优点是 20 dB 调整相关,这对于系统的增益控制很有用处。

基准控制放大器的小信号带宽大约是 1.4 MHz,在 COMP1 和 AVcc 之间连接一个外部电容可减小带宽。控制放大器的输出 COMP1 通过一个 50 pF 的电容在内部进行补偿,该电容限制了控制放大器的小信号带宽,并减少了它的输出阻抗。任何附加的外部电容进一步限制了带宽,而且作为一个滤波器减少了来自基准放大器的噪声分布。

要得到任何重构波形的最优失真性能,只需加入一个 0.1 μ F 外部电容就能实现。如果 I_e 固定,那么建议使用一个 0.1 μ F 的陶瓷电容。同样,因为控制放大器对小功率运行是最优的,所以对需要大信号幅度的复合应用而言,应该考虑用一个外部控制放大器来增强大信号的复合带宽。

固定 Rset 后,有双电源和单电源两种系统使得 I_e 变化。在单电源系统中,内部基准被截止了,而 REFIO 的共模电压在 0.1~1.25 V 之间变化,REFIO 能够用单一电源放大器或 DAC 来驱动,这样就允许在固定 Rset 时 I_e 变化,因为 REFIO 的输入阻抗大约是 1M Ω ,因此可以在电压模式拓扑结构中使用一个简化的低成本 R-2R 阶梯 DAC 电路来控制增益。

3) 模拟输出

AD9764 产生两个互补电流输出 IOUTA 和 IOUTB,可以作单端或差分结构。对要求直线线性最优的应用来说, I_a 和 I_b 应通过运放的虚地接法(I-V 变换)把输出电流保持在虚拟地上,使得 AD9764 的输出阻抗被固定,从而可明显地减少它对线性的影响。

在差分或单端输出结构中,IOUTA 和 IOUTB 降低了电压输出,减少了信号和它的输出阻抗之间的相关性,这样就提高了失真性能。即使 IOUTA 和 IOUTB 的电压范围从 -1.0 V 延伸到 +1.25 V,但是最优失真性能是在 IOUTA 和 IOUTB 上的最大满量程信号大约不超过 0.5 V 时才能得到。因此,应用上应适当地选择一个以地为中心抽头的变压器,允许 AD9764 为不同的负载提供所需要的功率和电压电平,同时保持 IOUTA 和 IOUTB 上较低电压幅度。当要求一差分或单输出结构的直流耦合应用时则要选取适当的负载电阻。

4) 数字输入

AD9764 的数字输入包括 14 位数据和一个时钟输入。这 14 位并行数据输入遵循标准二进制编码,其中, D13 是最高有效位(MSB)而 D0 是最低有效位(LSB)。当所有数据位都是逻辑 1 时, IOUTA 产生一个满量程输出电流, IOUTB 产生一个满量程电流的互补输出,而这两个输出成为输入码的一个函数。

AD9764 用一个边沿触发主从锁存器来完成数字接口。在时钟的上升沿后更新 DAC 输出,如图 3-29 所示,而且该输出可以支持一个高达 125 MHz 的时钟速率。该时钟能够在任何占空比下工作,只要该占空比满足额定的锁存脉冲宽度。只要满足了额定的最小倍数,即使这些传送边沿位置可能影响数字馈通和失真性能,当输入数据传送在一个 50% 的占空比时钟的下降沿时,一般也能够实现最佳性能。

AD9764 的内部数字电路能够工作在 2.7~5.5 V 的数字电源范围内。因此, 当 DV_{CC} 成为调节 TTL 驱动器的最高电平因素时, 数字输入同样能够调节 TTL 电平。

因为 AD9764 具有 125 MHz 的更新能力, 在实现最优性能时, 时钟和数据输入信号的质量就显得非常重要。AD9764 工作在较低的逻辑幅度和相应的数字电源上时, 将有较低的数据馈通干扰和片内数字噪声。数据接口电路的驱动器应满足 AD9764 的最小建立和保持时间, 同样要求满足它的最小/最大输入逻辑电平门限。



图 3-29 时序关系

数字信号路径应尽量短, 其长度应避免传播延时失配。在 AD9764 数字输入和驱动器输出之间插入一个低值的电阻网络(20~100 Ω), 有助于减少在数字输入上的任何超调和瞬变, 这些超调和瞬变是数据馈通所引起的。对更长的导线长度和高数据率, 应该考虑传送技术加上适当的端电阻来保持“干净”的数字输入。

外部时钟驱动器电路应给 AD9764 提供一个满足最小/最大逻辑电平的低起伏时钟输入, 同时提供快速边沿。快速时钟边沿将有助于减小任何起伏, 这些起伏在重构波形上体现为相位噪声。

5) 休眠模式操作

AD9764 具有掉电功能。在额定电源范围 2.7~5.5 V 和额定温度范围内, 掉电功能截止输出电流并使电源电流小于 8.5 mA。在休眠模式下可通过加一个逻辑电平“1”到 SLEEP 引脚上来激活它。该数字输入同样包括一个激活下拉电路, 它确保了 AD9764 的输入没有连接时能够工作。

AD9764 的掉电和启动特性取决于连接到 COMP1 的保持电容的值。通常, 对于 0.1 μF 电容值, AD9764 掉电只要 5 μs , 重新启动大约要 3.25 ms。注意, 在用外部控制放大器时, 不应该使用休眠模式。

3.7.2 应用电路与编程

对需要最优动态性能的应用来说, 建议用差分输出结构。差分输出结构可以包括一个 RF 变压器或一个差分运放。变压器提供了最优的高频性能, 并可用在任何交流耦合的场合。差分运放适用于需要交流耦合、一个双极输出和信号增益可变等应用场合。

单端输出适用于要求单极性电压输出的场合, 如果 IOUTA 或 IOUTB 连接到一个以 AGND 为参考地的负载电阻上, 就会得到一个正单极性输出电压。它更适用于需要直流耦合和以地为参考的输出电压的单电源系统。同样, 一个放大器能够由一个 I-V 转换器来组成, 这样就把 IOUTA 或 IOUTB 转换成一个负单极性电压。该结构提供了最佳直线性度, 因为 IOUTA 或 IOUTB 被保持在一个虚拟地上。注意, IOUTA 提供的性能比 IOUTB 稍微好一些。

1. 利用变压器的差分耦合

RF 变压器能用来进行差分到单端信号的转换,如图 3-30(a)所示。差分耦合变压器输出的信号有最优失真性能,其谱含量位于变压器通带内,并具有传送两倍功率到负载的能力。具有不同阻抗比的变压器同样可以用于阻抗匹配。注意,此时变压器只提供交流耦合。

在变压器的初级绕组上的中心抽头必须被连接到 AGND 上,从而为 IOUTA 和 IOUTB 提供必要的直流通路。出现在 IOUTA 和 IOUTB 上的互补电压(V_a 和 V_b)对称地在 AGND 附近摆动,并保持在 AD9764 的额定输出范围内。在这些应用中,变压器的输出通过一个无源重构滤波器或电缆连接到负载上。

2. 利用运放作差动输出

运放能用来进行差分到单端的转换,如图 3-30(b)所示。AD9764 用两个相等的 $25\ \Omega$ 负载电阻来取电压。由 IOUTA 和 IOUTB 所提供的差分电压通过差分运放转换成单端信号。一个可选电容放在 IOUTA 和 IOUTB 之间,形成低通滤波器的一个实极点。该电容还可以增强运放的失真性能。

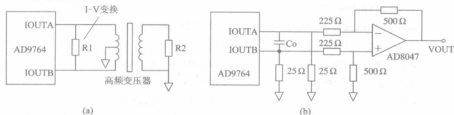


图 3-30 差分耦合输出电路

(a) 利用变压器的差分耦合; (b) 用运放作直流差分输出

通常,该结构的共模抑制由电阻匹配来确定。在该电路中,使用了 AD8047 的差分运放电路来提供某些附加信号增益。运放必须工作在双电源上,因为它的输出大约是 $\pm 1.0\text{ V}$ 。高速放大器具有保护 AD9764 差分性能的能力,同时它还满足其他要求(如成本、功率等),所以应选择高速放大器。在优化该电路时应该考虑所用的差分增益,该运放的增益规定了电阻值和满量程输出摆幅。

图 3-31 是由 STC89C54 构成的单极性 D/A 转换电路。其中, C_1 与 R_1 构成了低通滤波网络, R_2 是运放的平衡电阻。输出转换电压表达式是:

$$V_o = -R_1 \times I_o$$

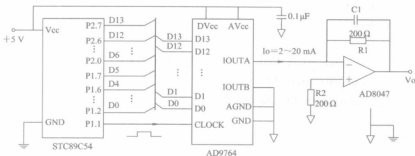


图 3-31 单极性 D/A 转换电路

3. 编程方法

对于图 3-31 所示的接口方法, 用 C51 实现的源程序如下:

```
#include <reg52.h>

s bit  CLOCK=P1^1;          /*定义时钟引脚*/

void delay(unsigned char x)  /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

/*功能为向 AD9764 写入一个整型数据, 参数 data_2 是写入的数据*/
void WriteAD9764(unsigned int data_2)
{
    unsigned char dat_h, dat_l;
    unsigned int data_16;
    data_16=data_2 & 0x3fff;    /*处理数据*/
    data_16=data_16<<2;        /*向左移 2 位*/
    dat_h= data_16/256;        /*取出(D13~D6)的高 8 位*/
    dat_l= data_16%256;        /*取出(D5~D0)的低 6 位*/
    P2=dat_h;                  /*送高 8 位数据到器件*/
    P1=dat_l;                  /*送低 6 位数据到器件*/
    CLOCK=1;                   /*产生上升沿*/
    delay(0x5);                /*延时*/
    CLOCK=0;                   /*产生下降沿*/
    delay(0x5);                /*延时*/
}

void main(void)               /*这是一个产生锯齿波的测试程序*/
{
    unsigned int da_data;
    CLOCK=0;                   /*CLOCK=0(低电平)*/
    da_data=0;
    while(1)
    {
        WriteAD9764(da_data); /*写入 14 位数据*/
        da_data=da_data+0x10;
        if(da_data>=0x3fff) da_data=0;
    }
}
```

第4章 高性能串行存储器的使用与编程

4.1 M25Pxx 大容量低成本 SPI 总线存储器

4.1.1 硬件与功能描述

M25Pxx 是一个 512 kb~32 Mb 带有先进写保护逻辑,采用高速 SPI 总线访问的串行快闪存储器。存储器由多个扇区组成,每个扇区包含 256 或 128 页。每页 256 字节,每页可一次写 1~256 字节。可以利用芯片擦除指令擦除整个存储器,或利用扇区擦除指令一次擦除一个扇区。

该器件可广泛应用于税控收款机、便携式仪表、GPS 导航、智能仪器、消费类产品和单片机应用系统中。

1. 主要性能特点

- (1) 系列器件闪存容量: 512 kb~32 Mb;
- (2) 编程时间: 每页(256 字节)的典型值为 1.4 ms;
- (3) 擦除种类: 扇区擦除(256 kb 或 512 kb)和整片擦除(512 kb~32 Mb);
- (4) 接口: 采用 SPI 总线兼容的串行接口;
- (5) 时钟速率: 40~50 MHz;
- (6) 单电源供电范围: 2.7~3.6 V(标准 3.3 V);
- (7) 功耗: 深度节电模式下,功耗的典型值为 1 μ A;
- (8) 擦写次数: 每扇区超过 100 000 次/编程周期;
- (9) 保存时间: 数据保存能力超过 20 年;
- (10) 工作温度范围: -40~+85℃。

2. 内部结构与引脚说明

M25Pxx 的内部结构与引脚排列如图 4-1 所示。

M25Pxx 系列器件的引脚排列分 SOP-8 与 SOP-16 两种,分别如图 4-1(b)、(c)所示。

其中:

- (1) $\overline{\text{HOLD}}$ 是 SPI 接口数据保持端,低电平有效。在低电平期间,数据传输被暂停,输出口呈现高阻,输入信号被忽略。当变为高电平时,方可继续传输。
- (2) $\overline{\text{W}}$ 是写保护端,低电平有效,在传输数据时,必须保持高电平。
- (3) $\overline{\text{S}}$ 是 SPI 接口片选端,低电平有效,在高电平期间,数据输出口呈现高阻状态。

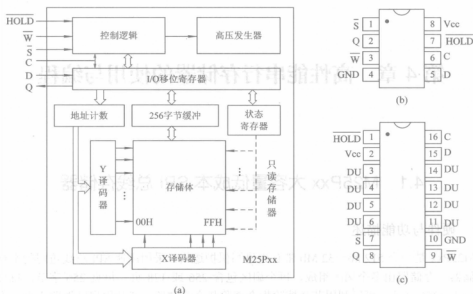


图 4-1 M25Pxx 的内部结构与引脚排列

(a) 内部结构; (b)、(c) 引脚排列

(4) C 是 SPI 接口时钟输入端。

(5) D 是 SPI 接口数据输入端, 该数据可以是存储器的命令、地址和被编程的数据, 在时钟的上升沿将数据输入器件。

(6) Q 是 SPI 接口数据输出端, 在时钟的下降沿将数据送出。

(7) Vcc 是电源输入端。

(8) GND 是参考地。

(9) DU 是内部悬空, 未用。

M25Pxx 系列器件型号有 7 种: M25P32(32 Mb)、M25P16(16 Mb)、M25P80(8 Mb)、M25P40(4 Mb)、M25P20(2 Mb)、M25P10(1 Mb)和 P25P05(512 kb)。

每一个器件的结构大同小异, 主要差别在于存储空间不同。例如, M25P80 有 1 048 576 字节(1 MB); 16 个扇区, 每个扇区 64 KB; 4096 页, 每页 256 字节。表 4-1 是 M25P80 的存储组织。

表 4-1 M25P80 的存储组织

扇区	地址范围	
15	F0000H~FFFFFH	983 040~1 048 575
14	E0000H~EFFFFH	917 504~983 039
13	D0000H~DFFFFH	851 968~917 503
12	C0000H~CFFFFH	786 432~851 967
11	B0000H~BFFFFH	720 896~786 431
10	A0000H~AFFFFH	655 360~720 895

续表

扇区	地址范围	
9	90000H~9FFFFH	589 824~655 359
8	80000H~8FFFFH	524 288~589 823
7	70000H~7FFFFH	458 752~524 287
6	60000H~6FFFFH	393 216~458 751
5	50000H~5FFFFH	327 680~393 215
4	40000H~4FFFFH	262 144~327 679
3	30000H~3FFFFH	196 608~262 143
2	20000H~2FFFFH	131 072~196 607
1	10000H~1FFFFH	65 536~131 071
0	00000H~0FFFFH	0~65 535

3. 操作指令

M25Pxx 系列器件在读/写操作时分为写允许、写禁用、页写(每次写 256 字节)、扇区擦除、整片擦除、字节读取、掉电操作等方式。每一次操作都有各自的操作指令,如表 4-2 所示。

表 4-2 M25Pxx 的操作指令集

指令	描述	1 字节指令码	地址字节	伪字节	数据字节
WREN	写允许	0000 0110	06H	0	0
WRDI	写禁用	0000 0100	04H	0	0
RDID	读取标识	1001 1111	9FH	0	0
RDSR	读取状态寄存器	0000 0101	05H	0	0
WRSR	写状态寄存器	0000 0001	01H	0	0
READ	读字节数据	0000 0011	03H	3	0
FAST_READ	高速读取字节数据	0000 1011	0BH	3	1
PP	页写	0000 0010	02H	3	0
SE	扇区擦除	1101 1000	D8H	3	0
BE	整片擦除	1100 0111	C7H	0	0
DP	深度节电	1011 1001	B9H	0	0
RES	退出节电并读签名	1010 1011	ABH	0	3
	退出深度节电			0	0

4. 操作时序

M25Pxx 的操作时序如图 4-2 所示。数据(D)的输入在时钟的上升沿锁入,数据(Q)的输出在时钟的下降沿输出。



图 4-2 M25Pxx 的操作时序

5. 状态寄存器的格式

M25Pxx 的状态寄存器的格式如表 4-3 所示。

表 4-3 M25Pxx 的状态寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
SRWD	0	0	BP2	BP1	BP0	WEL	WIP

其中:

- (1) WIP 位是“写”进行中标志,当器件正忙(正在“擦除”或正在“写入数据”)时, WIP = 1, 当不忙(“擦除”或“写入”结束)时, WIP = 0。
- (2) WEL 位是写使能标志, 当该位置“1”时, 器件内部允许“擦除”或“写入数据”; 当该位为“0”时, 将禁止写入或擦除芯片。
- (3) BP2~BP0 是块(扇区)保护设置位, 如表 4-4 所示。
- (4) SRWD 位是状态寄存器写保护位。当为“1”和 \overline{W} 信号为低电平时, 将保护数据的写入。

表 4-4 M25P80 BP2~BP0 位的设置

状态寄存器的位内容			存储区块内容	
BP2	BP1	BP0	保护区域(扇区)	未保护区域(扇区)
0	0	0	无	0~15
0	0	1	第 15 扇区(1 个)	0~14
0	1	0	第 14、15 扇区(2 个)	0~13
0	1	1	第 12、13、14、15 扇区(4 个)	0~11
1	0	0	第 8~15 扇区(8 个)	0~7
1	0	1	第 0~15 扇区(16 个)	无
1	1	0	第 0~15 扇区(16 个)	无
1	1	1	第 0~15 扇区(16 个)	无

6. M25Pxx 的操作过程

1) 写允许指令(WREN)

M25Pxx 写允许指令码是 06H, 向器件输入图 4-3 所示的时序, 就可以对器件进行“写”或“擦除”操作。器件是否完成“处理”, 可查看状态寄存器的 WIP 位(WIP = 0 时结束)。

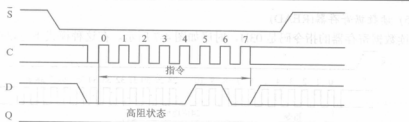


图 4-3 写允许指令时序

2) 写操作禁用指令(WRDI)

M25Pxx 写操作禁用指令码是 04H, 向器件输入图 4-4 所示的时序, 就禁止了对器件的任何写操作(包括擦除操作)。

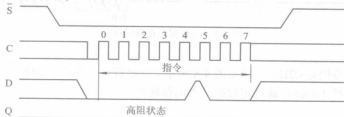


图 4-4 写禁用指令时序

3) 读取状态寄存器(RDSR)

读取状态寄存器的指令码是 05H, 格式如表 4-3 所示, 其指令如图 4-5 所示。

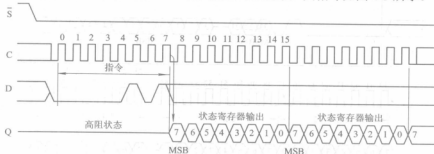


图 4-5 读状态寄存器时序

4) 写状态寄存器(WRSR)

写状态寄存器的指令码是 01H, 时序如图 4-6 所示, 格式见表 4-3。

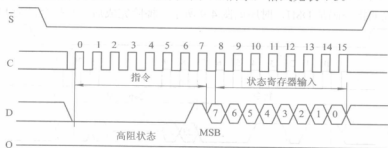


图 4-6 写状态寄存器时序

5) 读数据寄存器(READ)

读数据寄存器的指令码是 03H, 时序如图 4-7 所示。在这种模式下可以连续读出数据。

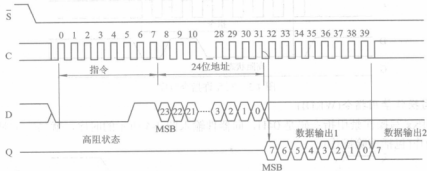


图 4-7 读数据寄存器时序

6) 页写(PP)

页数据写指令码是 02H, 时序如图 4-8 所示。该过程可一次连写 256 字节。在 \bar{S} 的上升沿开始, 大约等待 1.4 ms, 就可将数据写入闪存地址。

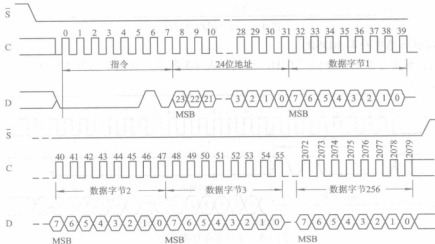


图 4-8 页写数据时序

7) 扇区擦除(SE)

扇区擦除的指令码是 D8H, 时序如图 4-9 所示。擦除完成后, 数据全部为 FFH。

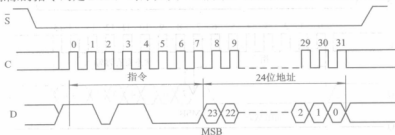


图 4-9 扇区擦除时序

8) 整片擦除(BE)

整片擦除指令码是 C7H, 时序如图 4-10 所示。擦除完成后, 数据全部为 FFH。

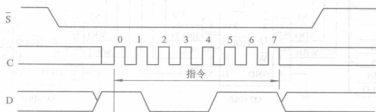


图 4-10 整片擦除时序

9) 休眠掉电操作(DP)

休眠掉电操作指令码是 B9H, 时序如图 4-11 所示。

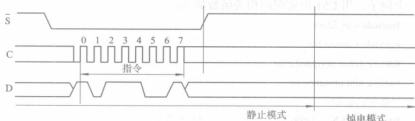


图 4-11 休眠掉电时序

10) 退出深度节电方式(RES)

退出深度节电指令码是 ABH, 时序如图 4-12 所示。

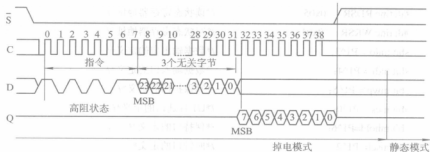


图 4-12 退出深度节电时序

4.1.2 应用电路与编程

1. 应用电路

STC89C54 单片机与 M25P80 闪存连接电路如图 4-13 所示。其中, U1 的片选接单片机的 P1.0, U2 的片选接单片机的 P1.1, 其他 SCL、SDO、SDA、WBH 和 HOLD 信号分别接单片机的 P1.2~P1.6。这种连接方法可在总线上挂接 SPI 总线的多片器件。

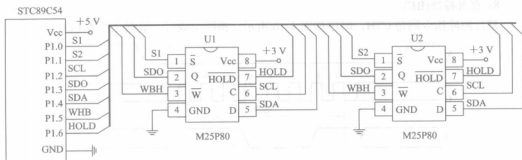


图 4-13 M25P80 与单片机的基本连接

2. 编程方法

作为一个例子，用 C51 所实现的相关函数如下：

```
#include <reg52.h>
#include <intrins.h>
#define uchar unsigned char
#define uint unsigned int
/*命令字定义*/
#define WREN    0x06      /*写允许*/
#define WRDI    0x04      /*写禁止*/
#define BE      0xc7      /*整片擦除*/
#define RES2    0xab      /*唤醒*/
#define DP      0xb9      /*休眠*/
#define RDSR    0x05      /*读状态寄存器地址*/
#define WRSR    0x01      /*写状态寄存器地址*/
sbit mdo = P1^3;          /*数据输出端的定义*/
sbit mdi = P1^4;          /*数据输入端的定义*/
sbit mwp = P1^5;          /*写保护口的定义*/
sbit mcs = P1^0;          /*U1 片选口的定义*/
sbit mhold = P1^6;        /*保持口的定义*/
sbit mscl = P1^2;         /*时钟口的定义*/
sbit mcs2 = P1^1;         /*U2 片选口的定义*/
uchar xdata wrbuffer[256]; /*写缓冲区*/
uchar sr=0;               /*全局变量*/

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}
```

/*发字节函数，注意这里并没有片选，在调用时根据实际情况在函数外进行片选设置*/

```

void sendbyte(uchar sedata)    /*发 8 位数据函数*/
{
    uchar i;
    for(i=0; i<8; i++)
    {
        mdi=(bit)(sedata & 0x80);
        sedata<<=1;
        msc1=1; _nop_();
        msc1=0; _nop_();
    }
}

uchar receivebyte(void)    /*接收 8 位数据函数*/
{
    uchar redata=0;
    uchar i;
    for(i=0; i<8; i++)
    {
        redata<<=1;
        if(mdo==1)
            redata++;    /*可用 redata=redata|0x01 代替*/
        msc1=1; _nop_();
        msc1=0; _nop_();
    }
    return(redata);
}

/*适用于允许写、写保护、全擦除、进入省电模式和退出省电模式等命令*/
void com mAnd(uchar secmd)    /*单字节命令
{
    mcs=0;    /*片选=0*/
    delay(10);
    sendbyte(secmd);    /*发送命令*/
    delay(5);
    mcs=1;    /*片选=1*/
}

uchar rdsr(void)    /*读状态寄存器命令*/
{
    uchar ch=0;
    mcs=0;    /*片选=0*/
    delay(10);
    sendbyte(RDSR);    /*发送命令(RDSR)0x05*/
    ch=receivebyte();    /*读数据, 值在 ch 中*/
    delay(5);
    mcs=1;    /*片选=1*/
    return ch;
}

```



```

void wrsr(uchar sestr)           /*写状态寄存器命令*/
{
    mcs=0;                       /*片选=0*/
    delay(10);                   /*延时*/
    sendbyte(WRSR);              /*写状态寄存器命令 0x01*/
    sendbyte(sesr);              /*状态寄存器命令*/
    delay(5);                   /*片选=1*/
    mcs=1;
}

void wait_m80(void)              /*等待 flash 完成工作*/
{
    while(rdsr() & 0x01 == 0x01); /*当 sr=1 时未结束, 当 sr=0 时已结束*/
}

void se(uchar sector)           /*扇区擦除函数*/
{
    com mAnd(WREN);              /*写允许*/
    delay(5); mcs=0;             /*片选=0*/
    delay(10);
    sendbyte(0xd8);              /*扇区擦除*/
    sendbyte(sector);            /*扇区(24 位)*/
    sendbyte(0);                 /*地址*/
    sendbyte(0);                 /*地址*/
    delay(5); mcs=1;             /*片选=1*/
}

void flashInit(void)            /*闪存初始化*/
{
    mwp=1;                       /*不写保护*/
    mcs=1; mcs2=1;
    mhold=1;                     /*不采用数据保持*/
    com mAnd(RES2);              /*唤醒存储器 0xab*/
    delay(10);
    wrsr(0);                     /*清状态寄存器的各位*/
}

/*可以读任意地址的数据[慢读 flash], 共 16 个扇区, 每扇区 256 页, 每页 256 字节*/
void read(uchar sector, uchar page, uchar addr)
{
    uint i;
    mcs=0;                       /*片选=0*/
    delay(10);
    sendbyte(0x03);              /*读数据命令*/
    sendbyte(sector);            /*扇区*/
    sendbyte(page);              /*page=0~255*/

```

```

sendbyte(addr);          /*addr=0~255*/
for(i=0; i<256; i++)
    wrbuffer[i]=receivebyte();    /*这样写一次读一页数据(256 字节)*/
delay(5);
mcs=1;                    /*片选=1*/
}

void pp(uchar sector, uchar page)    /*页写命令*/
{
    uint i;
    com mAnd(WREN);          /*写允许*/
    delay(5);
    mcs=0;                    /*片选=0*/
    delay(10);
    sendbyte(0x2);            /*页写命令*/
    sendbyte(sector);          /*写扇区*/
    sendbyte(page);            /*写页*/
    sendbyte(0);                /*地址(页内地址)*/
    for(i=0; i<256; i++)
        sendbyte(wrbuffer[i]);    /*写入 256 个数据*/
    delay(5);
    mcs=1;                    /*片选=1，等待约 1.5 ms 写入 256 个数据*/
}

unsigned char xdata xdataA[512];    /*定义的数据区*/

void main(void)                /*主函数*/
{
    unsigned int i;
    for(i=0; i<256; i++)
    {
        xdataA[i]=i; xdataA[256+i]=i;    /*产生的模拟数据 0, 1, 2, 3, ...*/
        flashInit();                    /*初始化 flash*/
        se(0);                            /*擦除 0 扇区*/
        wait_m80();                        /*查询等待擦除完成*/
        for(i=0; i<256; i++)
            wrbuffer[i]=xdataA[i];
        pp(0, 0);                        /*把 wrbuffer 里的数据写到 0 扇区和 0 页地址中*/
        wait_m80();                        /*查询完成*/
        for(i=256; i<512; i++)
            wrbuffer[i-256]=xdataA[i];
        pp(0, 1);                        /*写 0 扇区和 1 页数据*/
        wait_m80();                        /*查询完成*/
        for(i=0; i<256; i++)
            wrbuffer[i]=0;                /*清除缓存的数据*/
    }
}

```

```

read(0, 0, 0);          /*读取 0 扇区 0 页 256 个数据存入 wrbuffer[]中*/
wait_m80();             /*查询完成*/
for(i=0; i<256; i++)
    xdataA[i]=wrbuffer[i];    /*存入 256 个数据*/
for(i=0; i<256; i++)
    wrbuffer[i]=0;           /*清除缓存的数据*/
read(0, 1, 0);          /*读取 0 扇区 1 页 256 个数据存入 wrbuffer[]中*/
wait_m80();             /*查询完成*/
for(i=256; i<512; i++)
    xdataA[i]=wrbuffer[i];    /*存入 256 个数据*/
while(1);
}

```

4.2 EN25B32 高速大容量低成本 SPI 总线存储器

4.2.1 硬件与功能描述

EN25B32 是一个 32 Mb(4 MB)的串行快闪存储器。内置保护单元具有高速 SPI 总线访问能力。该器件在页指令控制下,一次编程多达 256 个字节。EN25B32 有 68 个扇区(其中,63 个扇区每扇区是 64 KB、一个扇区是 32 KB、一个扇区是 16 KB、一个扇区是 8 KB、两个扇区每扇区是 4 KB,共 4096 KB)。可以利用芯片擦除指令擦除整个存储器,或利用扇区擦除指令一次擦除任意扇区。

该器件可广泛应用于税控收款机、便携式仪表、GPS 导航、智能仪器、消费类产品和单片机应用系统中。

1. 主要性能特点

- (1) 器件闪存容量: 32 Mb(4096 KB, 16 384 页);
- (2) 编程时间: 字节编程时间为 7 μ s(典型值), 每页(256 字节)的典型值为 1.5 ms;
- (3) 擦除时间: 扇区擦除为 300~800 ms, 整片擦除为 25 s(典型值);
- (4) 接口: 采用 SPI 总线兼容的串行接口;
- (5) 时钟速率: 100 MHz(典型值);
- (6) 单电源供电范围: 2.7~3.6 V(标准 3.3 V);
- (7) 功耗: 正常工作典型值为 5 mA, 节电模式下典型值为 1 μ A;
- (8) 擦写次数: 每扇区超过 10 万次/编程周期;
- (9) 保存时间: 数据保存能力超过 20 年;
- (10) 工作温度范围: 商业级 0~+70 $^{\circ}$ C, 工业级 -40~+85 $^{\circ}$ C。

2. 内部结构与引脚说明

EN25B32 的内部结构与引脚排列如图 4-14 所示。

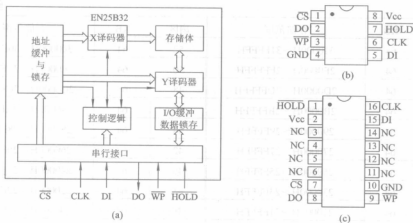


图 4-14 EN25B32 的内部结构与引脚排列

(a) 内部结构; (b)、(c) 引脚排列

EN25B32 闪存器件的引脚排列有 SOP-8 与 SOP-16 两种, 分别如图 4-14(b)、(c)所示。其中:

- (1) $\overline{\text{HOLD}}$ 是 SPI 接口数据保持端, 低电平有效。
- (2) $\overline{\text{WP}}$ 是写保护端, 低电平有效, 在传输数据时, 必须保持高电平。
- (3) $\overline{\text{CS}}$ 是 SPI 接口片选端, 低电平有效, 在高电平期间, 数据输出口呈现高阻状态。
- (4) CLK 是 SPI 接口时钟输入端。
- (5) DI 是 SPI 接口数据输入端, 该数据可以是存储器的命令、地址和被编程的数据, 在时钟的上升沿将数据输入器件。
- (6) DO 是 SPI 接口数据输出端, 在时钟的下降沿将数据送出。
- (7) Vcc 是电源输入端。
- (8) GND 是参考地。
- (9) NC 是内部悬空, 未用。

3. EN25B32 的扇区空间分配

EN25B32 的总存储空间为 4 194 304 B(4 MB), 分为 68 个扇区, 每个扇区的空间不是均等分配的, 如表 4-5 所示。

表 4-5 EN25B32 的扇区分配

扇区	容量/KB	存储范围	扇区	容量/KB	存储范围
67	64	3F0000H~3FFFFFFH	66	64	3E0000H~3FFFFFFH
65	64	3D0000H~3DFFFFFFH	64	64	3C0000H~3CFFFFFFH
63	64	3B0000H~3BFFFFFFH	62	64	3A0000H~3AFFFFFFH
61	64	390000H~39FFFFFFH	60	64	380000H~38FFFFFFH
59	64	370000H~37FFFFFFH	58	64	360000H~36FFFFFFH
57	64	350000H~35FFFFFFH	56	64	340000H~34FFFFFFH
55	64	330000H~33FFFFFFH	54	64	320000H~32FFFFFFH

续表

扇区	KB	存储范围	扇区	KB	存储范围
53	64	310000H~31FFFFH	52	64	300000H~30FFFFH
51	64	2F0000H~2FFFFFH	50	64	2E0000H~2EFFFFH
49	64	2D0000H~2DFFFFH	48	64	2C0000H~2CFFFFH
47	64	2B0000H~2BFFFFH	46	64	2A0000H~2AFFFFH
45	64	290000H~29FFFFH	44	64	280000H~28FFFFH
43	64	270000H~27FFFFH	42	64	260000H~26FFFFH
41	64	250000H~25FFFFH	40	64	240000H~24FFFFH
39	64	230000H~23FFFFH	38	64	220000H~22FFFFH
37	64	210000H~21FFFFH	36	64	200000H~20FFFFH
35	64	1F0000H~1FFFFFH	34	64	1E0000H~1EFFFFH
33	64	1D0000H~1DFFFFH	32	64	1C0000H~1CFFFFH
31	64	1B0000H~1BFFFFH	30	64	1A0000H~1AFFFFH
29	64	190000H~19FFFFH	28	64	180000H~18FFFFH
27	64	170000H~17FFFFH	26	64	160000H~16FFFFH
25	64	150000H~15FFFFH	24	64	140000H~14FFFFH
23	64	130000H~13FFFFH	22	64	120000H~12FFFFH
21	64	110000H~11FFFFH	20	64	100000H~10FFFFH
19	64	0F0000H~0FFFFFH	18	64	0E0000H~0EFFFFH
17	64	0D0000H~0DFFFFH	16	64	0C0000H~0CFFFFH
15	64	0B0000H~0BFFFFH	14	64	0A0000H~0AFFFFH
13	64	090000H~09FFFFH	12	64	080000H~08FFFFH
11	64	070000H~07FFFFH	10	64	060000H~06FFFFH
9	64	050000H~05FFFFH	8	64	040000H~04FFFFH
7	64	030000H~03FFFFH	6	64	020000H~02FFFFH
5	64	010000H~01FFFFH	4	32	008000H~00FFFFH
3	16	004000H~007FFFH	2	8	002000H~003FFFH
1	4	001000H~001FFFH	0	4	000000H~000FFFH

4. 状态寄存器的格式

EN25B32 的状态寄存器的格式如表 4-6 所示。

表 4-6 EN25B32 的状态寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
SRP	0	0	BP2	BP1	BP0	WEL	BUSY

其中:

(1) BUSY 位是器件“忙”标志位,当器件正在“擦除”或正在写入数据时, BUSY = 1;

当擦除或写入结束时, $BUSY = 0$ 。用该位的状态可以查询器件是否完成目前的操作。

(2) WEL 位是写使能标志, 当该位置“1”时, 器件内部允许“擦除”或“写入数据”; 当该位为“0”时, 将禁止写入或擦除芯片。

(3) $BP2 \sim BP0$ 是块(扇区)保护设置位, 如表 4-7 所示。

(4) SRP 位是状态寄存器写保护位。当该位为“1”和 \overline{WP} 信号为低电平时, 将保护数据的写入。

表 4-7 EN25B32 $BP2 \sim BP0$ 位的设置

状态寄存器的位内容			存储器块内容	
$BP2$	$BP1$	$BP0$	保护区域(扇区)	未保护区域(扇区)
0	0	0	无	0~67
0	0	1	扇区 0, 000000H~000FFFFH(4 KB)	1~67
0	1	0	扇区 0~1, 000000H~001FFFFH(8 KB)	2~67
0	1	1	扇区 0~2, 000000H~003FFFFH(16 KB)	3~67
1	0	0	扇区 0~3, 000000H~007FFFFH(32 KB)	4~67
1	0	1	扇区 0~4, 000000H~00FFFFFFH(64 KB)	5~67
1	1	0	扇区 0~35, 000000H~1FFFFFFFH(2048 KB)	36~67
1	1	1	所有	无

5. 操作指令

EN25B32 的操作指令如表 4-8 所示。

表 4-8 EN25B32 的操作指令

含 义	指令名	字节1	字节2	字节3	字节4	字节5	字节6	字节n
写允许	WREN	06H						
写禁止	WRDI	04H						
读状态寄存器	RDSR	05H	D7~D0					
写状态寄存器	WRSR	01H	D7~D0					
读数据	READ	03H	A23~A16	A15~A8	A7~A0	D7~D0	下一数	继续
快读数据	F_READ	0BH	A23~A16	A15~A8	A7~A0	空操作	D7~D0	下一数
页写数据	PP	02H	A23~A16	A15~A8	A7~A0	D7~D0	下一数	继续
扇区擦除	SE	D8H	A23~A16	A15~A8	A7~A0			
整片擦除	BE	C7H						
省电模式	DP	B9H						
唤醒掉电状态	RDI	ABH	空操作	空操作	空操作	D7~D0		
读制造ID信息	RID	90H	空操作	空操作	00H	M7~M0	I7~I0	
读器件ID	RDID	9FH	M7~M0	I15~I8	I7~I0			
进入OTP模式	OTP	3AH						

表中, $A23 \sim A0$ 是地址, $D7 \sim D0$ 是数据, $M7 \sim M0$ 是读出的制造信息, $I15 \sim I0$ 是 ID 信息。

6. EN25B32 的操作时序

对于 EN25B32 的操作既可以通过标准的 SPI 模式工作(时序见图 4-15), 也可以通过单片机的 I/O 口模拟 SPI 总线时序。模拟时将 $\overline{\text{CS}}$ 、DI、DO、SCK、 $\overline{\text{HOLD}}$ 和 $\overline{\text{WP}}$ 六个信号线分别接入单片机的 I/O 口, 然后根据 EN25B32 的操作指令按图 4-15 所示的时序模拟。数据的输入(DI)在时钟(SCK)的上升沿将数据锁入, 数据的输出(DO)在时钟的下降沿送出。

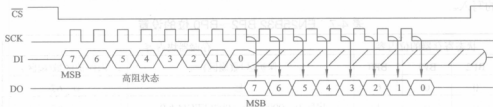


图 4-15 SPI 总线时序

表 4-8 中的功能操作过程可参考图 4-3~图 4-12 所示的时序。

4.2.2 应用电路与编程

1. 应用电路

EN25B32 与单片机的连接如图 4-16 所示。其中, $\overline{\text{HOLD}}$ 和 $\overline{\text{WP}}$ 均接电源。

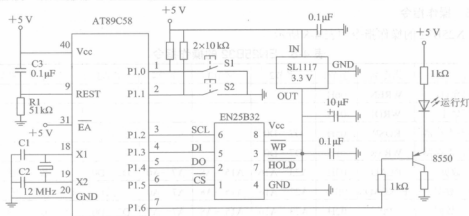


图 4-16 基本连接电路

2. 编程方法

EN25B32 的软件编程与 4.1.2 节 M25Pxx 器件的编程方法基本相同。这里通过 C51 只编写保护“扇区 0~1(8 KB)”和读出器件 ID 号的主函数, 子函数用 4.1.2 节的对应程序。

```
sbit mdo=P1^4;           /*数据输出口的定义*/
sbit mdi=P1^3;           /*数据输入口的定义*/
sbit mcs=P1^5;           /*片选口的定义*/
sbit mscl=P1^2;          /*时钟口的定义*/
unsigned char xdata xdataA[512]; /*定义的数据区*/
```

```

void main(void)
{
    unsigned int i;
    unsigned char md, id[2];
    for(i=0; i<256; i++)
    { xdataA[i]=i; xdataA[i+256]=i; } /*产生的模拟数据*/
    flashInit(); /*初始化 flash*/
    se(0); /*擦除 0 扇区*/
    wait_m80(); /*查询等待擦除完成*/
    se(1); /*擦除 1 扇区*/
    wait_m80(); /*查询等待擦除完成*/
    for(i=0; i<256; i++)
        wrbuffer[i]=xdataA[i];
    pp(0, 0); /*模拟写入 0 扇区和 0 页数据*/
    wait_m80(); /*等待完成*/
    for(i=256; i<512; i++)
        wrbuffer[i-256]=xdataA[i];
    pp(1, 0); /*模拟写入 1 扇区和 0 页数据*/
    wait_m80(); /*等待完成*/
    wsr(0x0A); /*写状态寄存器 0AH, 将 0~1 扇区保护*/
    wait_m80(); /*等待完成*/
    mcs=0; /*片选=0*/
    delay(10);
    sendbyte(0x8F); /*发送读 ID 号指令*/
    md=receivebyte(); /*读出制造信息*/
    id[0]=receivebyte(); /*读出 ID15~ID8*/
    id[1]=receivebyte(); /*读出 ID7~ID0*/
    delay(5);
    mcs=1;
    while(1);
}

```

4.3 FM25xx 系列高速高性能铁电 SPI 总线存储器

4.3.1 硬件与功能描述

FM25xx 系列器件是一款 256 B~256 KB 的非易失性存储器,采用先进的铁电技术制造,又名 FRAM, 但该器件执行读和写操作与 RAM 很相似。FM25xx 提供 45 年的数据保存时间,同时消除了由 E²PROM 和其他非易失性存储器导致的复杂性和可靠性问题。

FM25xx 系列器件与串行 E²PROM 不同, 该系列器件以总线速度进行写操作, 无需延时。下一个总线周期可以立即开始, 无需进行数据轮询。另外, 该器件真正提供了无限次的写入次数, 而且, 因为写操作不需要内部的提升电路, 所以在写操作过程中, FRAM 比 E²PROM 消耗的功率要低得多。

FM25xx 系列器件的这些功能使得它非常适合用在需要频繁或快速写操作的非易失性存储器中。该器件采用了高速的 SPI 总线协议, 在传输速度方面加强了 FRAM 技术的高速写性能, 确保可靠地工作在 $-40\sim+85^{\circ}\text{C}$ 的工业温度范围内。在器件封装上, FM25xx 系列器件可和大多数 SPI 总线的 E²PROM 串行器件引脚兼容。

1. 主要性能特点

- (1) 系列器件存储容量: 256 B~256 KB;
- (2) 读/写次数: 无限次;
- (3) 数据保留时间: >40 年;
- (4) 接口: 采用高达 20 MHz 的 SPI 总线串行接口;
- (5) 兼容性: 硬件上可直接替换 SPI 总线的 E²PROM;
- (6) 完整的写保护机制: 硬件保护或软件保护;
- (7) 单电源供电范围: 2.7~3.65 V(3.3 V 器件), 4.5~5.5 V(5 V 器件);
- (8) 功耗: 正常工作时的典型值为 5 mA, 待机电流的典型值为 1 μA ;
- (9) 工作温度范围: $-40\sim+85^{\circ}\text{C}$ 或 $-40\sim+125^{\circ}\text{C}$ 。

2. 内部结构与引脚说明

FM25xx 系列器件的内部结构与引脚排列如图 4-17 所示。

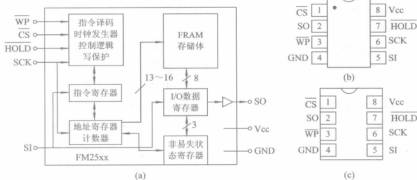


图 4-17 FM25xx 的内部结构与引脚排列

(a) 内部结构; (b)、(c) 引脚排列

FM25xx 系列器件的引脚排列有 SOP-8 与 TDFN-8 两种, 分别如图 4-17(b)、(c)所示。其中:

(1) $\overline{\text{HOLD}}$ 是 SPI 接口数据保持端, 当主 CPU 必须中断存储器当前的操作而执行另一个任务时, 可以使用 $\overline{\text{HOLD}}$ 管脚。当 $\overline{\text{HOLD}}$ 管脚为低电平时, 当前操作被挂起。器件忽略 SCK 或者 $\overline{\text{CS}}$ 上的任何跳变。

(2) $\overline{\text{WP}}$ 是写保护端, 该管脚为低电平时对状态寄存器进行写保护。因为其他的写保护

特征由状态寄存器控制, 所以这个管脚的作用很重要。注意, M25CL64 的 $\overline{\text{WP}}$ 管脚的功能与 FM25040 的不同, FM25040 的 $\overline{\text{WP}}$ 禁止器件的所有写操作。在传输数据时, $\overline{\text{WP}}$ 管脚必须保持高电平。

(3) $\overline{\text{CS}}$ 是 SPI 接口片选端, 该管脚为低电平时, 才可操作器件。当该管脚为高电平时, 器件进入低功耗的待机模式, 忽略其他输入的状态, 而且所有的输出处于三态。当该管脚为低电平时, 器件内部激活 $\overline{\text{SCK}}$ 信号。 $\overline{\text{CS}}$ 的下降沿必须在每个操作码之前出现。

(4) $\overline{\text{SCK}}$ 是 SPI 接口时钟输入端, 所有的 I/O 活动都是与串行时钟同步的。输入在时钟的上升沿被锁存, 输出在下降沿输出。因为器件是静态的, 所以时钟频率可以是 0~20 MHz 之间的任意值且随时都可以被中断。

(5) SI 是 SPI 接口数据输入端, 该数据可以是存储器的命令、地址和被编程的数据, 在时钟的上升沿将数据输入器件。

(6) SO 是 SPI 接口数据输出端, 在时钟的下降沿将数据送出。

(7) V_{CC} 是电源输入端。

(8) GND 是参考地。

FM25xx 系列器件有 +5 V 供电的, 如 FM25C02(256 B)~FM25C512(64 KB), 也有 3.3 V 供电的, 如 FM25L02(256 B)~FM25H20(256 KB)。其型号如表 4-9 所示。

表 4-9 FM25xx 型号一览表

型 号	容 量	最大电流/mA	最高速度/MHz	封 装	电 源
FM25H20	2 Mb	10	40	TDFN-8	3
FM25L512	512 kb	12	20	TDFN-8	3
FM25L256B	256 kb	10	20	SOP8, TDFN-8	3
FM25L256	256 kb	10	20	SOP8, TDFN-8	2.7~3.6
FM25256B	256 kb	15	20	SOP-8	5
FM25CL64-GA	64 kb	7	16	SOP-8	3~3.6
FM25CL64	64 kb	10	20	SOP-8, TDFN-8	2.7~3.6
FM25640-GA	64 kb	2.7	4	SOP-8	4.5~5.5
FM25640	64 kb	3	5	SOP-8	5
FM25L16	16 kb	5.5	18	SOP-8, TDFN-8	2.7~3.6
FM25C160-GA	16 kb	6.5	15	SOP-8	5
FM25C160	16 kb	8	20	SOP-8	5
FM25040A-GA	4 kb	6	14	SOP-8	5
FM25L04	4 kb	6	14	SOP-8, TDFN-8	2.7~3.6
FM25L04-GA	4 kb	2.2	10	SOP-8	3.0~3.6
FM25040A	4 kb	8	20	SOP-8	5

3. 外围接口

FM25xx 采用串行外围接口总线(SPI)。它的工作速度被指定为高达 20 MHz。这种高速串行总线为主微控制器提供了高性能的串行通信。大多数普通的微控制器都有硬件 SPI 端口, 允许直接连接。对于没有 SPI 端口的微控制器, 使用普通的 I/O 口引脚来仿真 SPI 端口也是十分方便的。FM25xx 的工作模式有两种: SPI 模式 0 和模式 3。SPI 接口一共使用 4 个

管脚：时钟、数据输入、数据输出和片选。典型系统配置使用一个或者多个 FM25xx 器件，微控制器提供专用的 SPI 端口，如图 4-18(a)所示。请注意，所有器件的时钟、数据输入和数据输出管脚是相同的。每个 FM25xx 器件的片选和保持管脚必须分开驱动。对于不含有专用 SPI 总线的微控制器，可以使用通用 I/O 端口。为了减少控制器上的硬件资源，也可将两个数据管脚(SI, SO)连接在一起，并且可以将 HOLD 管脚接高电平，如图 4-18(b)所示。

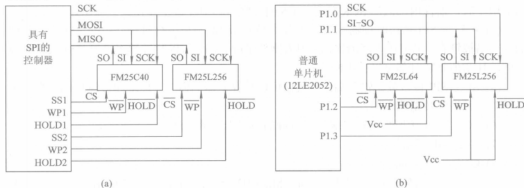


图 4-18 FM25xx 的外围接口

(a) 用专用的 SPI 接口；(b) 用普通的 I/O 模拟 SPI 口

SPI 接口是一个同步串行接口，它使用时钟和数据管脚，用于支持挂在总线上的多个器件。每个器件都可以由片选激活，一旦总线被主控器激活了片选，FM25xx 将开始监控时钟和数据线。 $\overline{\text{CS}}$ 下降沿、时钟和数据之间的关系由 SPI 模式控制。器件在每个片选的下降沿确定 SPI 模式，一共有四种模式，但 FM25xx 只支持模式 0 和 3。图 4-19 显示了模式 0 和 3 请求的信号关系。对于这两种模式，数据在 SCK 的上升沿时移入到 FM25xx，并且器件会在 $\overline{\text{CS}}$ 生效之后的第一个上升沿等待数据。如果时钟的起始状态是高电平，则它在开始传输数据之前将下降以产生第一个上升沿。SPI 协议由操作码控制，这些操作码指定了器件的命令。 $\overline{\text{CS}}$ 生效之后，总线主控器传输的第一个字节是操作(指令)码。传输操作码之后，可以传输任何地址和数据。

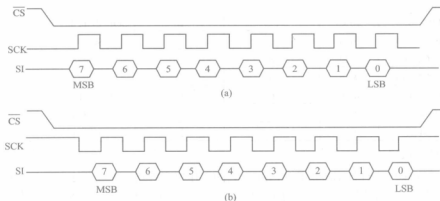


图 4-19 SPI 模式 0 和模式 3

(a) SPI 模式 0: CPOL=0, CPHA=0; (b) SPI 模式 3: CPOL=1, CPHA=1

4. 操作指令

FM25xx 有 6 种操作码的指令, 它们由总线主控器发送给 FM25xx, 如表 4-10 所示。这些操作码控制了存储器执行的功能。它们可以分为三类: 第一类是不带数据的操作指令, 它们执行单一的功能, 例如使能写操作; 第二类是带一个字节指令, 写入或者读出, 它们对状态寄存器进行操作; 第三类包括了执行存储器操作的命令, 这些命令后面跟随有地址和一个或多个数据字节。

表 4-10 FM25xx 的操作指令集

操作名称	含 义	操 作 码	字节数
WREN	设置写使能锁存器	0000 0110B(06H)	0
WRDI	写禁止	0000 0100B(04H)	0
RDSR	读状态寄存器	0000 0101B(05H)	1 字节
WRSR	写状态寄存器	0000 0001B(01H)	1 字节
READ	读存储器数据	0000 0011B(03H)	N 字节
WRITE	写存储器数据	0000 0010B(02H)	N 字节

1) 设置写使能锁存器(WREN)

FM25xx 上电后的状态是禁止写操作。WREN(06H)指令必须在其他写操作之前发送。WREN 操作码的发送将允许用户对器件进行“状态寄存器”和“数据存储器”的写操作。WREN 操作码的发送将导致内部写使能锁存器置位。状态寄存器的标志位 WEL 表明锁存器的状态。WEL = 1 表示进行写操作是允许的。任何写操作的结束都将使得写使能锁存器自动清零(WEL = 0), 这样就防止了在没有 WREN 指令时进行更多的写操作。写允许时序如图 4-20 所示。

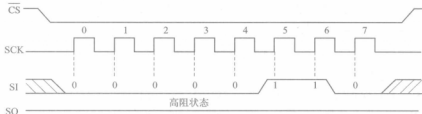


图 4-20 写允许时序

2) 写禁止(WRDI)

WRDI(04H)指令通过清除写使能锁存器来禁止所有的写操作。用户通过读取状态寄存器中的 WEL 位来验证写操作是禁止的(即 WEL = 0)。写禁止时序如图 4-21 所示。

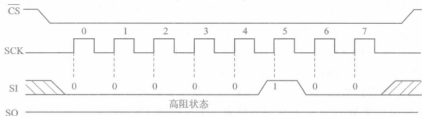


图 4-21 写禁止时序

3) 读状态寄存器(RDSR)

RDSR(05H)指令允许总线主控器对状态寄存器的内容进行验证。读取状态寄存器能够获得写保护特性的当前信息。紧跟 RDSR 操作码之后, FM25xx 将返回带有状态寄存器内容的一个字节, 时序如图 4-22 所示。

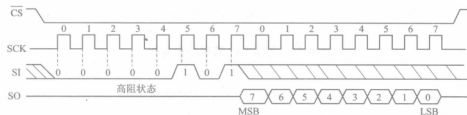


图 4-22 读状态寄存器时序

4) 写状态寄存器(WRSR)

WRSR(01H)指令允许向状态寄存器写入一个字节, 这个字节的内容允许用户选择某些写保护特性。在发送 WRSR 指令之前, $\overline{\text{WP}}$ 管脚必须为高电平状态或者无效状态。注意, 对于 25CL64 的 $\overline{\text{WP}}$ 管脚, 只是对状态寄存器进行写保护, 而不是写保护存储阵列。在发送 WRSR 指令之前, 用户必须发送一个 WREN 指令允许写操作。WRSR 指令所执行的就是一个写操作, 这可清除写使能锁存器的某些位, 时序如图 4-23 所示。

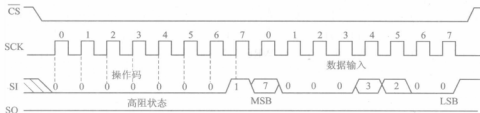


图 4-23 写状态寄存器时序

5) 状态寄存器与写保护

FM25xx 的写保护特性是多样的。第一, WREN 操作码必须在进行任何写操作之前发送。假定使用 WREN 指令使能了写操作, 则存储器进行的写操作由状态寄存器控制。像上面描述的那样, 使用 WRSR 指令就可以写状态寄存器。状态寄存器的格式如表 4-11 所示。

表 4-11 状态寄存器的格式

位	D7	D6	D5	D4	D3	D2	D1	D0
名称	WPEN	0	0	0	BP1	BP0	WEL	0

位 D0 和 D4~D6 的值是固定的, 并且不能修改(注意, E²PROM 中的有些位在此处已经不再需要, 因为 FRAM 是以实时的速度进行写操作的, 永远不会处于忙状态)。WPEN、BP1 和 BP0 控制写保护的特性。它们是非易失性的。WEL 标志表明写使能锁存器的状态。直接向状态寄存器的 WEL 位写入“0”或“1”是不会影响它的状态的。也就是说, 该位只能由 WREN 命令使 WEL = 1, 并可通过结束一个写周期(CS 为高电平)或使用 WRDI 命令来

清零(WEL=0)。BP1 和 BP0 是存储器块写保护位, 它们指定了受到写保护的存储区域, 如表 4-12 所示。

表 4-12 存储器块写保护设置

BP1	BP0	保护地址范围
0	0	无保护
0	1	保护高 1/4 位置的存储器空间(如 FM25L512 的保护空间是 6000H~7FFFH)
1	0	保护高 1/2 位置的存储器空间(如 FM25L512 的保护空间是 4000H~7FFFH)
1	1	保护全部的存储器空间(如 FM25L512 的保护空间是 0000H~7FFFH)

BP1、BP0 位和写使能锁存器是禁止存储器进行写操作的唯一方法。WPEN 位控制硬件 $\overline{\text{WP}}$ 管脚的作用。当 WPEN 为低电平时, $\overline{\text{WP}}$ 管脚被忽略; 当 WPEN 为高电平时, $\overline{\text{WP}}$ 管脚控制对状态寄存器的写访问; 当 WPEN=1 和 $\overline{\text{WP}}=0$ 时, 状态寄存器被硬件写保护。

该方法提供了一个写保护机制, 这使得软件在任何环境中都不能向存储器写入。如果 BP1 和 BP0 都设置为 1, WPEN 位也设置为 1, $\overline{\text{WP}}$ 设置为 0, 则会发生写保护。因为块保护位将存储器设置为写保护, 硬件 $\overline{\text{WP}}$ 管脚信号防止块保护位的更改(如果 WPEN 为高电平), 所以发生了不能向存储器写入的情况。因此在该状态中, 写保护的许可必须有硬件参与。表 4-13 给出了写保护条件。

表 4-13 写保护

WEL	WPEN	$\overline{\text{WP}}$	受保护的存储块	未受保护的存储块	状态寄存器
0	x	x	被保护	被保护	被保护
1	0	x	被保护	未被保护	未被保护
1	1	0	被保护	未被保护	被保护
1	1	1	被保护	未被保护	未被保护

6) 存储器写操作(WRITE)

SPI 接口有很高的时钟频率, 这就更能体现 FRAM 的快速写性能。与使用 SPI 总线 E²PROM 不同, FM25xx 以总线速度执行连续的写操作。它不需要使用页寄存器, 可以执行任何数量的连续写操作。

存储器进行的所有写操作都是从 WREN 操作码开始的, 紧跟着的是一个操作码 WRITE。WRITE(02H)操作码之后是两个字节地址值(注意, 如果所用器件的存储空间小于 256 B, 地址就只有一个字节; 如果空间大于 64 KB, 则地址需要三个字节)。这是写操作的第一个数据字节的开始地址, 随后的字节都是数据字节, 它们是连续写入的。只要总线主控器继续发送时钟信号并且保持 $\overline{\text{CS}}$ 为低电平, 地址就会内部递增。如果达到最后地址 IFFFF(FM25CL64), 地址计数器将跳转至 0000H。数据以 MSB 在前的方式写入。 $\overline{\text{CS}}$ 的上升沿中止一个 WRITE 操作。写操作时序如图 4-24 所示。E²PROM 使用页缓冲来增加它们的写吞吐量, 这使得该技术固有的缓慢的写操作得到了改善。在时钟的驱动作用下(第 8 个时钟之后), 每个字节立即写入到 FRAM 阵列, 所以 FRAM 存储器不需要页缓冲, 即允许写入任何数量的字节而无需页缓冲延迟。

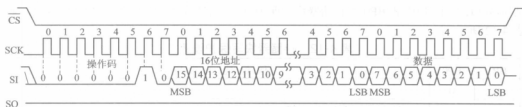


图 4-24 写时序

7) 存储器读操作(READ)

在 $\overline{\text{CS}}$ 的下降沿之后, 总线主控器发布一个 READ(03H)操作码。READ 指令之后是一个双字节地址值。这是读操作的首字节的起始地址(注意, 如果所用器件的存储空间小于 256 字节, 则地址就只有一个字节)。在发送操作码和地址之后, 器件在主机发送时钟时读取数据, SI 输入在读取数据字节期间被忽略, 随后的数据字节被连续读取。只要总线主控器继续发送时钟信号并且保持 $\overline{\text{CS}}$ 为低电平, 地址将内部递增。如果达到最后地址 FFFFH(FM25L512), 则地址计数器将跳转至 0000H。数据是以 MSB 在前的方式读出的。 $\overline{\text{CS}}$ 的上升沿可终止一个 READ 操作。读操作时序如图 4-25 所示。

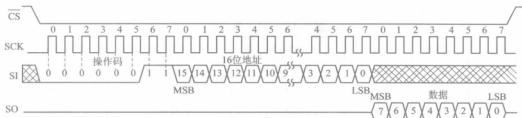


图 4-25 读时序

4.3.2 应用电路与编程

按图 4-18(b)所示的电路连接, 用 C51 实现的完整函数如下:

```
#include <reg52.h>
#include <intrins.h>

#define WREN 0x06    /*写允许指令*/
#define WRDI 0x04    /*写禁止指令*/
#define RDSR 0x05    /*读状态寄存器指令*/
#define WRSR 0x01    /*写状态寄存器指令*/
#define READ 0x03    /*读数据指令*/
#define WRITE 0x02    /*写数据指令*/

sbit mdo=P1^1;       /*定义 SPI 输出口*/
sbit mdi=P1^1;       /*定义 SPI 输入口*/
sbit mcs1=P1^2;      /*FM25L64 的片选*/
```

```

sbit mcs2=P1^3;           /*FM25L256 的片选*/
sbit msc1=P1^0;           /*定义 SPI 时钟口*/
unsigned char xdata wrbuffer[256]; /*写缓冲区*/
unsigned char sr=0;        /*全局变量*/
void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

/*发送字节函数，在调用时根据实际情况在函数外进行 mcs(片选)设置*/
void sendbyte(unsigned char sedata) /*发送 8 位数据函数*/
{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        mdi=(bit)(sedata & 0x80); /*先发送高位*/
        sedata<<=1;
        msc1=1; _nop_();
        msc1=0; _nop_();
    }
}

unsigned char receivebyte(void) /*接收 8 位数据函数*/
{
    unsigned char redata=0, i;
    for(i=0; i<8; i++)
    {
        redata<<=1;
        if(mdo==1) redata++;
        msc1=1; _nop_();
        msc1=0; _nop_();
    }
    return(redata);
}

/*适用于写允许、写禁止的操作*/
void send_cmd(unsigned char secmd) /*单字节命令*/
{
    delay(10);
    sendbyte(secmd); /*发送命令*/
    delay(5);
}

```



```

void send_cmd_cs1(unsigned char secmd) /*操作 cs1 单字节命令*/
{
    mcs1=0; /*使 FM25L64 的 cs=0*/
    send_cmd(secmd); /*发送指令*/
    mcs1=1; /*使 FM25L64 的 cs=1*/
}

void send_cmd_cs2(unsigned char secmd) /*操作 cs2 单字节命令*/
{
    mcs2=0; /*使 FM25L256 的 cs=0*/
    send_cmd(secmd); /*发送指令*/
    mcs2=1; /*使 FM25L256 的 cs=1*/
}

void rdsr_cs1(void) /*读状态寄存器命令*/
{
    mcs1=0; /*使 FM25L64 的 cs=0*/
    delay(10);
    sendbyte(RDSR); /*发送命令(RDSR)0x05*/
    sr=receivebyte(); /*读数据, 值在 sr 中*/
    delay(5);
    mcs1=1; /*使 FM25L64 的 cs=1*/
}

void rdsr_cs2(void) /*读状态寄存器命令*/
{
    mcs2=0; /*使 FM25L256 的 cs=0*/
    delay(10);
    sendbyte(RDSR); /*发送命令(RDSR)0x05*/
    sr=receivebyte(); /*读数据, 值在 sr 中*/
    delay(5);
    mcs2=1; /*使 FM25L256 的 cs=1*/
}

void wrsr_cs1(unsigned char sesr) /*写状态寄存器命令*/
{
    mcs1=0;
    delay(10);
    sendbyte(WRSR); /*写状态寄存器命令 0x01*/
    sendbyte(sesr); /*状态寄存器*/
    delay(5);
    mcs1=1;
}

```

```

void wrsr_cs2(unsigned char sestr)          /*写状态寄存器命令*/
{
    mcs2=0;
    delay(10);
    sendbyte(WRSR);                        /*写状态寄存器命令 0x01*/
    sendbyte(sestr);                       /*状态寄存器*/
    delay(5);
    mcs2=1;
}

void flashInit(void) /*FM25xx 初始化*/
{
    mcs1=1;
    mcs2=1;
}

void read_cs1(unsigned int address, unsigned int numb) /*可以读任意地址的数据*/
{
    unsigned int i;
    unsigned char abh, abl;
    abh=address/256;                       /*取出高 8 位地址*/
    abl=address%256;                       /*取出低 8 位地址*/
    mcs1=0;                                /*使 FM25L64 的 cs=0*/
    delay(10);
    sendbyte(READ);                        /*读数据命令*/
    sendbyte(abh);                         /*送高 8 位地址*/
    sendbyte(abl);                         /*送低 8 位地址*/
    for(i=0; i<numb; i++)
        wrbuffer[i]=receivebyte();        /*读 numb 个数据*/
    delay(5);
    mcs1=1;                                /*使 FM25L64 的 cs=1*/
}

void read_cs2(unsigned int address, unsigned int numb) /*可以读任意地址的数据*/
{
    unsigned int i;
    unsigned char abh, abl;
    abh=address/256;                       /*取出高 8 位地址*/
    abl=address%256;                       /*取出低 8 位地址*/
    mcs2=0;                                /*使 FM25L256 的 cs=0*/
    delay(10);
    sendbyte(READ);                        /*读数据命令*/

```

```

        sendbyte(abh);          /*送高 8 位地址*/
        sendbyte(abl);          /*送低 8 位地址*/
        for(i=0; i<numb; i++)
            wrbuffer[i]=receivebyte();    /*读 numb 个数据*/
        delay(5);
        mcs2=1;                  /*使 FM25L256 的 cs=1*/
    }

    /*给 FM25L64 任意地址写入任意个数据*/
void write_cs1(unsigned int address, unsigned int numb)    /*写任意数据函数*/
{
    unsigned int i;
    unsigned char abh, abl;
    abh=address/256;            /*取出高 8 位地址*/
    abl=address%256;            /*取出低 8 位地址*/
    send_cmd_cs1(WREN);         /*写允许*/
    delay(5);
    mcs1=0;                     /*使 FM25L64 的 cs=0*/
    delay(10);
    sendbyte(WRITE);            /*写指令*/
    sendbyte(abh);               /*高 8 位地址*/
    sendbyte(abl);               /*低 8 位地址*/
    for(i=0; i<numb; i++)
        sendbyte(wrbuffer[i]);  /*写入 numb 个数据*/
    delay(5);
    mcs1=1;                     /*使 FM25L64 的 cs=1*/
}

    /*给 FM25L256 任意地址写入任意个数据*/
void write_cs2(unsigned int address, unsigned int numb)    /*写任意数据函数*/
{
    unsigned int i;
    unsigned char abh, abl;
    abh=address/256;            /*取出高 8 位地址*/
    abl=address%256;            /*取出低 8 位地址*/
    send_cmd_cs2(WREN);         /*写允许*/
    delay(5);
    mcs2=0;                     /*使 FM25L256 的 cs=0*/
    delay(10);
    sendbyte(WRITE);            /*写指令*/
    sendbyte(abh);               /*高 8 位地址*/

```

```

        sendbyte(abl);           /*低 8 位地址*/
        for(i=0; i<numb; i++)
            sendbyte(wrbuffer[i]); /*写入 numb 个数*/
        delay(5);
        mcs2=1;                  /*使 FM25L256 的 cs=1*/
    }
    unsigned char xdata xdataA[256]; /*定义的数据区*/
    void main(void)
    { unsigned int i;
      for(i=0; i<256; i++)
          xdataA[i]=i;           /*产生模拟数据*/
      flashInit();               /*初始化 flash*/
      for(i=0; i<256; i++)
          wrbuffer[i]=xdataA[i]; /*向写入缓冲器送数*/
      write_cs1(0x200, 256);      /*向 FM25L64 的 200H~200H+256 连续送数据*/
      wrsr_cs1(0x8e);             /*数据全部被保护, 要再写入数据必须先送入 0x8e*/
      write_cs2(0x1000, 256);     /*向 FM25L256 的 1000H~1000H+256 连续送数据*/
      wrsr_cs2(0x8e);             /*数据全部被保护, 要再写入数据必须先送入 0x8e*/
      read_cs1(0x200, 256);       /*从 FM25L64 中连续读出 256 个数*/
      read_cs2(0x1000, 256);      /*从 FM25L256 中连续读出 256 个数*/
      while(1);                  /*结束*/
    }

```

4.4 FM24xx 系列高速高性能铁电 I²C 总线存储器

4.4.1 硬件与功能描述

FM24xx 系列器件是一款 256 B~64 KB 的非易失性存储器, 它采用先进的铁电处理技术, 其执行过程就像普通 RAM 的一样, 保存数据时间高达几十年, 同时消除了由 E²PROM 和其他非易失性存储器导致的复杂性和可靠性问题。

FM24xx 以总线速度进行写操作, 下个总线周期可以立即开始, 不需要延时。另外, 该系列产品能够承受的写操作次数远远多于 E²PROM。而且, 因为写操作不需要内部的提升电路, 所以, 在写操作过程中, 铁电存储器比 E²PROM 消耗的功率要低得多。

该器件可广泛应用于高速存储并且需要掉电记忆的应用系统中。另外, 因引脚完全兼容大部分的 E²PROM, 所以可方便地直接替换绝大部分的 I²C 总线的存储器。

1. 主要性能特点

- (1) 系列器件存储容量: 256 B~64 KB;
- (2) 读/写次数: 无限次;

- (3) 数据掉电保留时间: >10 年;
- (4) 接口: 采用高达 1 MHz 的 I^2C 总线接口, 支持 100 kHz 或 400 kHz 时钟频率;
- (5) 兼容性: 硬件上可直接替换 I^2C 总线的 E^2PROM ;
- (6) 保护机制: 采用硬件保护;
- (7) 单电源供电范围: $2.7\sim 3.6\text{ V}$ (3.3 V 器件), $4.5\sim 5.5\text{ V}$ (5 V 器件);
- (8) 功耗: 正常工作时典型值为 $75\text{ }\mu\text{A}$ (100 kHz), 待机电流的典型值为 $1\text{ }\mu\text{A}$;
- (9) 工作温度范围: $-40\sim +85^\circ\text{C}$ 。

2. 内部结构与引脚说明

FM24xx 系列器件的内部结构与引脚排列如图 4-26 所示。FM24xx 的内部地址为 256 B 单元 $\sim 64\text{ KB}$ 单元, 每个单元为 8 位, 数据位被串行移出, 采用 2 线协议, 包括一个从地址和扩展地址。

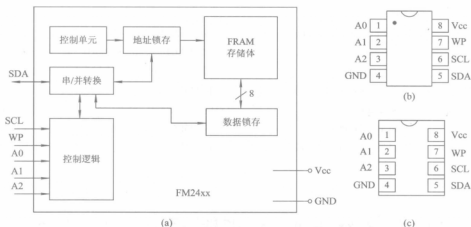


图 4-26 FM24xx 系列器件的内部结构与引脚排列

(a) 内部结构; (b)、(c) 引脚排列

FM24xx 系列器件的引脚排列有 SOP-8 与 TDFN-8 两种, 分别如图 4-26(b)、(c)所示。其中:

- (1) A0 \sim A2 是器件地址, 通过 A0 \sim A2 的编码, 可在总线上挂接 8 个 I^2C 器件;
- (2) WP 是写保护端, 高电平有效, 写入数据时, WP 要置低电平;
- (3) SCL 是 I^2C 接口的时钟输入端;
- (4) SDA 是 I^2C 接口的数据输入/输出端(内部开路), 通过 SDA 线可传送地址和数据, 在时钟的上升沿将数据输入器件;
- (5) Vcc 是电源输入端;
- (6) GND 是参考地。

FM25xx 系列器件有 $+5\text{ V}$ 供电的 FM24C04(512 B) \sim FM24C512(64 KB)和 3.3 V 供电的 FM24CL04(512 B) \sim FM24CL64(8 KB)多个器件。其型号如表 4-14 所示。

表 4-14 FM24xx 型号一览表

型 号	容 量/kb	最大电流/mA	最高速度/MHz	封 装	电 源/V
FM24C512	512	1.5	1	EIAJ SOP-8	5
FM24C256	256	1.2	1	EIAJ SOP-8	5
FM24CL64	64	0.4	1	SOP-8, TDFN-8	2.7~3.6
FM24C64	64	1.2	1	SOP-8	5
FM24CL16	16	0.4	1	SOP-8, TDFN-8	2.7~3.6
FM24C16A	16	1.0	1	SOP-8	5
FM24CL04	4	0.3	1	SOP-8	2.7~3.6
FM24C04A	4	1.0	1	SOP-8	5

3. 使用说明

FM24xx 器件支持双向数据传输握手协议, 这种协议允许通过一个简单的 2 线系统总线在各种设备之间进行通信操作。该 2 线总线被定义为串行数据线(SDA)和串行时钟线(SCL)。典型的总线结构如图 4-27 所示。



图 4-27 典型的总线结构

控制线路传输设备称为主设备(Master), 受控制的设备称为从设备(Slave)。在任何情况下 FM24xx 总是从设备, 因为 FM24xx 绝对不会发起一次数据传输。该总线最多可挂 8 个 FM24xx, 器件物理地址 A0~A2 必须按编码接到 Vcc 或 Vss 上。跟随在启动条件后, 主设备(发生器)必须发出“设备寻址字节(器件)”, 用于选择一个挂在系统总线上的从设备(接收器), 其中包括设备类别标志、设备地址、读或写操作标志等。然后, 接收器把数据线 SDA 置为“低”作为应答的确认信号(ACK)。确认信号(ACK)用于指示成功的数据传输。发送设备在发送 8 位数据后就释放数据总线, SDA 变高。在发送数据时每个时钟周期的下降沿传输一位数据。在第九个时钟周期, 接收器把数据线拉到“低”, 以此向发送器确认 8 位数据已被收到。总线操作时序如图 4-28 所示。

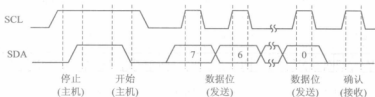


图 4-28 总线操作时序

1) 从器件的地址

从器件的硬件地址包括器件 ID 号、硬件地址 A2~A0 位及读/写位, 如表 4-15 所示。

表 4-15 从器件的硬件地址

器件 ID 号				硬件地址 A2~A0			读/写位
1	0	1	0	A2	A1	A0	R/ \overline{W}

2) 写过程

FM24xx 系列器件的单字节与多字节写($R/\overline{W}=0$)时序过程如图 4-29 所示。在写入过程中, 每写入一个字节(8 bit)就跟一个器件向主机的回答(A)信号。这一过程一定是:

开始条件 + 器件地址 + 回答位 + 存储地址 + 回答位 + 数据 + ... + 回答位 + 停止条件

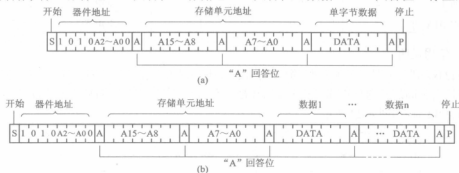


图 4-29 单字节与多字节写时序过程

(a) 单字节写; (b) 多字节写

3) 读过程

FM24xx 系列器件读数据分为读当前地址的单字节、多字节和选择某存储器地址。读数据($R/\overline{W}=1$)的过程如图 4-30 所示。

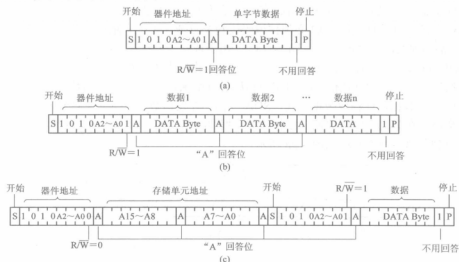


图 4-30 读数据过程

(a) 当前地址读; (b) 在当前地址下的多字节读; (c) 某地址下的随机读

4.4.2 应用电路与编程

1. 应用电路

四片 FM24CL64 与 STC12L4052 的典型接法如图 4-31 所示。其中, U1、U2、U3、U4 的器件地址分别是 0、1、2、3。数据线上的上拉电阻一般取 $5.1\sim 10\text{ k}\Omega$ 即可。在编程时要根据 FM24CL64 的硬件地址来完成每个器件的寻址。

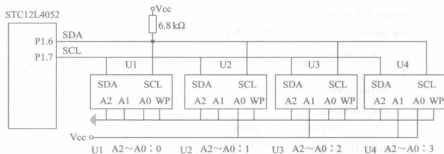


图 4-31 四片 FM24CL64 与 STC12L4052 的典型接法

2. 编程方法

根据图 4-31, 用 C51 实现的读/写函数如下:

```
#include <reg51.h>

sbit SCL=P1^6;          /*时钟端口*/
sbit SDA=P1^7;          /*数据端口*/

#define U1AB 0x00        /*U1 A2~A0 的硬件地址*/
#define U2AB 0x01        /*U2 A2~A0 的硬件地址*/
#define U3AB 0x02        /*U3 A2~A0 的硬件地址*/
#define U4AB 0x03        /*U4 A2~A0 的硬件地址*/

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for (i=0; i<x; i++);
}

void start_I2C(void)        /*启动函数*/
{
    SDA=1; SCL=1;          /*数据线=1, 时钟线=1*/
    delay(5);
    SDA=0; delay(5);
    SCL=0;
}

void stop_I2C(void)        /*停止函数*/
```



```

{
    SDA=0; SCL=0;          /*数据线=0, 时钟线=0*/
    delay(5);
    SCL=1; delay(5);        /*在 SCL=1 时, SDA 由 0 变 1*/
    SDA=1;
}

void send_8(unsigned char data8) /*写一个 8 位数据*/
{
    unsigned char dat, i;
    dat=data8;              /*发送数据*/
    for(i=0; i<8; i++)
    {
        if(dat&0x80) SDA=1;
        else SDA=0;
        delay(2); SCL=1;    /*时钟线=1*/
        delay(2); SCL=0;    /*时钟线=0*/
        delay(2); dat<<=1;  /*左移 1 位*/
    }
}

void read_ack(void)
{
    unsigned char ack;
    SDA=1;                  /*立即释放数据线*/
    /*接收 ACK*/
    delay(2); SCL=1;        /*第 9 个时钟*/
    if(SDA==0) ack=0;
    else ack=1;
    SCL=0;
}

void write_ack(void)        /*发送 ACK, 通知从机收到数据*/
{
    SCL=0; delay(2);
    SCL=1; delay(2);
    SDA=0;
    delay(2);
    SCL=0;
}

/*-----
功能为向 FM24xx 写入一个字节, byte 为写入的数据
-----*/
void Write_FM24xx_8(address, word, byte)

```

```

unsigned char address, byte;
unsigned int word;          /*存储地址*/
{
    unsigned char addh, addl;
    addh=word/256;          /*取出高 8 位地址*/
    addl=word%256;          /*取出低 8 位地址*/
    start_I2C();            /*启动*/
    send_8(0xa0 | address); /*发送器件从地址*/
    read_ack();             /*读 ACK*/
    send_8(addh);           /*发送从器件数据高位地址*/
    read_ack();             /*读 ACK*/
    send_8(addl);           /*发送从器件数据低位地址*/
    read_ack();             /*读 ACK*/
    send_8(byte);           /*发送数据*/
    read_ack();             /*读 ACK*/
    stop_I2C();             /*产生停止条件*/
}

void Write_FM24xx_n(address, word, byte, n) /*发送 n 个数据*/
unsigned char address, byte[];
unsigned int word;          /*存储地址*/
unsigned int n;             /*数据个数*/
{
    unsigned char addh, addl;
    unsigned int i;
    addh=word/256;          /*取出高 8 位地址*/
    addl=word%256;          /*取出低 8 位地址*/
    start_I2C();            /*启动*/
    send_8(0xa0 | address); /*发送器件从地址*/
    read_ack();             /*读 ACK*/
    send_8(addh);           /*发送从器件数据高位地址*/
    read_ack();             /*读 ACK*/
    send_8(addl);           /*发送从器件数据低位地址*/
    read_ack();             /*读 ACK*/
    for(i=0; i<n; i++)
    {
        send_8(byte[i]);    /*发送数据*/
        read_ack();         /*读 ACK*/
    }
    stop_I2C();             /*产生停止条件*/
}

```

```

}
/*-----
功能为从 FM24Cxx 读出一个字节数据, 参数 address 表示器件地址, word 表示存储地址
-----*/
unsigned char Read_FM24xx_8(address, word)
{
    unsigned char address;
    unsigned int word;
    {
        unsigned char j;
        unsigned char addh, addl, dat;
        addh=word/256;      /*取出高 8 位地址*/
        addl=word%256;      /*取出低 8 位地址*/
        start_I2C();        /*启动*/
        send_8(0xa0 | address); /*发送器件从地址*/
        read_ack();         /*读 ACK*/
        send_8(addh);        /*发送从器件数据高位地址*/
        read_ack();         /*读 ACK*/
        send_8(addl);        /*发送从器件数据低位地址*/
        read_ack();         /*读 ACK*/
        start_I2C();        /*启动*/
        send_8(0xa1 | address); /*发送器件从地址*/
        read_ack();         /*读 ACK*/
        for(j=0; j<8; j++)   /*读一个 8 位数据*/
        {
            dat<=1; delay(2);
            SCL=1; delay(2);
            if(SDA==0) dat &= 0xfe;
            else dat |= 0x01;
            SCL=0;
        }
        stop_I2C();         /*产生停止条件*/
        return(dat);        /*返回数据*/
    }
}
/*-----
功能为从 FM24Cxx 当前地址下读一个数据, 参数 address 为器件读地址
-----*/
unsigned char Read_FM24xx_1_8(unsigned char address)
{
    unsigned char dat, i;
    start_I2C();           /*启动*/

```

```

send_8(0xa1 | address);    /*发送器件从地址的读*/
read_ack();                /*读 ACK*/
for(i=0; i<8; i++)         /*读高 8 位数据*/
{
    dat <=<=1; delay(2);
    SCL=1; delay(2);
    if(SDA==0)    dat &= 0xfe;
    else    dat |= 0x01;
    SCL=0;
}
stop_I2C();                /*产生停止条件*/
return(dat);               /*返回数据*/
}

/*-----

```

功能为从 FM24Cxx 当前地址下读 n 个数据, 参数 address 为器件读地址, *p 为数据

-----*/

void Read_FM24xx_n_8(address, n, unsigned char *p)

unsigned char address;

unsigned int n;

{

unsigned char dat, i;

unsigned int j;

start_I2C(); /*启动*/

send_8(0xa1 | address); /*发送器件从地址的读*/

read_ack(); /*读 ACK*/

for(j=0; j<n-1; j++)

{ dat=0;

for(i=0; i<8; i++) /*读高 8 位数据*/

{

dat <=<=1; delay(2);

SCL=1; delay(2);

if(SDA==0) dat &= 0xfe;

else dat |= 0x01;

SCL=0;

}

*p=dat; /*将数据送入数组*/

p++;

write_ack(); /*产生 ACK*/

}

```

dat=0;
for(i=0; i<8; i++)
{
    dat <= 1; delay(2);
    SCL=1; delay(2);
    if(SDA==0) dat &= 0xfe;
    else dat |= 0x01;
    SCL=0;
}
*p=dat;
stop_I2C();
}

void main(void)
{
    unsigned char d1_data, d2_data[0x20], i;
    Write_FM24xx_8(U2AB, 0x1000, 0x88); /*给 U2 的 1000H 中写入 88H*/
    d1_data=Read_FM24xx_8(U2AB, 0x1000); /*将 U2 地址 1000H 数据读出*/
    for(i=0; i<0x20; i++)
        d2_data[i]=i; /*产生数据*/
    Write_FM24xx_n(U1AB, 0x0, d2_data, 0x20); /*给 U1 的 0~20H 中连续写入数据*/
    d1_data=Read_FM24xx_8(U1AB, 0);
    Read_FM24xx_n(U1AB, 0x20, d2_data); /*将 U1 的 0~20H 数据读出*/
    d1_data=Read_FM24xx_1_8(U3AB); /*读出 U3 当前的数据*/
    while(1); /*等待*/
}

```

4.5 X5643/45 带有 CPU 监视的 E²PROM 存储器

4.5.1 硬件与功能描述

X5643/45 系列器件把三种常用的功能——看门狗定时、电源电压监视和串行 E²PROM 组合在单个封装之内。这种组合降低了系统成本，减少了对电路板空间的要求并增加了可靠性。

看门狗定时器提供了独立的保护系统。当系统有故障时，在可选的超时周期之后，器件将以 RESET 或 RESET 信号做出响应。用户可以从三个预置的值中选择此周期。一旦选定，即使在电源周期变化之后，此周期也不改变。

利用器件的电源检测电路，可以保护用户系统使之免受电压状态的影响。当电源 V_{cc} 降到门限值以下时，系统复位。RESET 或 RESET 一直确保到电源 V_{cc} 返回到正常工作电压为止。

高电平至低电平的跳变。只要 $\overline{\text{CS}}/\text{WDI}$ 从高到低或从低到高跳变, 就都认为 $\overline{\text{RESET}}$ 或 RESET 输出正常。

(5) $\overline{\text{WP}}$ 是写保护端。当 $\overline{\text{WP}}$ 为低电平且非易失性位 WPEN 为“1”时, 向非易失性存储器写被禁止, 但是器件的其他功能仍正常; 当 $\overline{\text{WP}}$ 保持高电平时, 所有的功能包括对状态寄存器的非易失性写都正常。如果内部的状态寄存器写周期已经开始, 那么 $\overline{\text{WP}}$ 变为低电平时 WPEN 为“1”将不影响此写操作, 但在这些条件的后续将对状态寄存器的写操作禁止。当状态寄存器 WPEN 位为“0”时, $\overline{\text{WP}}$ 引脚的功能被禁止。这使得用户在 $\overline{\text{WP}}$ 引脚接地的情况下仍能对状态寄存器编程。当 WPEN 位被置成“1”时, $\overline{\text{WP}}$ 引脚的功能被允许(使能)。

(6) $\overline{\text{RESET}}$ 或 RESET 是复位输出脚。前者低电平有效, 后者高电平有效。其输出是漏极开路式的。只要 V_{CC} 下降至低于最小 V_{CC} 检测电平, 它便变为有效。它将保持有效直至 V_{CC} 上升到超过最小 V_{CC} 检测电平达 200ms 为止。如果允许看门狗定时器工作且 $\overline{\text{CS}}/\text{WDI}$ 保持高电平或低电平的时间长于所选的看门狗超时周期, 那么复位端也将变为有效。 $\overline{\text{CS}}/\text{WDI}$ 的下降沿将复位看门狗定时器。

3. 使用说明

X5643/45 可方便地与许多常用微控制器直接串行接口。该器件有两大用途: 第一个是用作 E^2PROM 8 KB 存储器; 第二个是用作 CPU 监视器。

1) 用作存储器

在用作存储器时, 与其他 SPI 总线的 E^2PROM 用法相似。该器件指令见表 4-16。地址及数据都以 MSB(最高有效位)在前的方式传送。SI 线上输入的数据在 $\overline{\text{CS}}$ 变为低电平之后的第一个 SCK 上升沿被采样。

表 4-16 指令集

指令名称	指令码	含 义
WREN	0000 0110	设置使能锁存器(允许写操作)
SFLB	0000 0000	设置标志位
WRDI/RFLB	0000 0100	复位写使能锁存器/复位标志位(禁止写)
RDSR	0000 0101	读状态寄存器的内容
WRSR	0000 0001	写状态寄存器(看门狗、块锁定、 WPEN 和标志位的设置)
READ	0000 0011	从所选地址的寄存器阵列读数据
WRITE	0000 0010	把数据写入开始于所选地址的寄存器阵列中

(1) 写使能锁存器。写使能锁存器在写操作之前必须进行设置。 WREN 指令可设置允许写入, 而 WRDI 指令将禁止(复位)写入。在上电和有效写周期完成之后, 此锁存器自动复位(禁止)。

(2) 状态寄存器。 RDSR 指令提供对状态寄存器的访问。在任何时候都可以读状态寄存器的内容, 即使在写周期也如此。状态寄存器的格式见表 4-17。

表 4-17 状态寄存器的格式

D7	D6	D5	D4	D3	D2	D1	D0
WPEN	FLAG	WD1	WD0	BL1	BL0	WEL	WIP

表中:

① WIP 位表示当前状态(即“正在写”)是只读位,它表示器件是否忙于内部非易失性写操作。利用 RDSR 指令读 WIP 位,可判断器件的状态。当此位被置为“1”时,非易失性写操作正在进行;当该位被置为“0”时,表示“不忙”。

② WEL 是“写使能锁存”位,表示“写使能”锁存器的状态。当设置为“1”时,锁存器置位;当设置为“0”时,锁存器复位。WEL 是只读位,它由 WREN 指令置位,由 WRDI 指令复位。

③ BL1 和 BL0 是“块锁定”设置位,即可设置 E²PROM 的保护级别。这些非易失性的位可用 WRSR 指令来编程并允许用户保护 E²PROM 阵列的 1/4、1/2、全部或都不保护。被锁定保护阵列的任何部分可以读出但不能写入,它们将一直保持在 BL1 和 BL0 位重新设置为止。块锁定方式见表 4-18。

表 4-18 块锁定方式

状态寄存器的位		被保护的 E ² PROM 阵列
BL1	BL0	X5643/X5645
0	0	全部未保护
0	1	保护: 1800H~1FFFH(高 2 KB)
1	0	保护: 1000H~1FFFH(高 4 KB)
1	1	保护: 0000H~1FFFH(全部 8 KB)

④ WD1 和 WD0 位是“看门狗定时器”定时时间选择位,用 WRSR 指令编程。定时时间的设置如表 4-19 所示。

表 4-19 定时时间的设置

状态寄存器的位		看门狗定时时间(超时)设置
WD1	WD0	典型值
0	0	1.4 s
0	1	600 ms
1	0	200 ms
1	1	禁止

⑤ FLAG 是只读标志位,表示易失性锁存器的状态,它可利用 SFLB 和 RFLB 指令由系统置位和复位。上电时,FLAG 位自动复位。

⑥ WPEN 是保护位。该位和 WP 引脚结合在一起可提供可编程的硬件写保护。当 WP 为低电平且 WPEN 位被编程为高电平时,所有的状态寄存器写操作均被禁止。

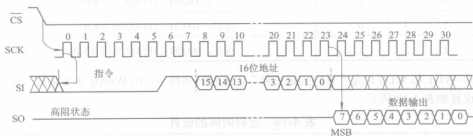
(3) 在线编程只读(ROM)方式。为了保护器件的某些状态,应避免偶然性因素导致的破坏。通过在硬件上把 WP 引脚接地,把所需的存储器部分写和块锁定为只读(ROM)方式,然

后将 WPEN 编程为高电平, 该方式可用于实现在线编程只读(ROM)功能。块保护组合见表 4-20。

表 4-20 块保护组合

WREN 命令	状态寄存器	器件引脚	存储块	存储块	状态寄存器
WEL	WPEN	$\overline{\text{WP}}$	被保护块	未被保护块	WPEN, BL1, BL0, WD1, WD0
0	x	x	被保护	被保护	被保护
1	1	0	被保护	未保护	被保护
1	0	x	被保护	未保护	未保护
1	x	1	被保护	未保护	未保护

(4) 读数据。当从 E²PROM 存储器阵列读数据时, 首先把 $\overline{\text{CS}}$ 拉至低电平以选择芯片。8 位的 READ 指令被发送到器件, 其后是 16 位的地址。在发送了 READ 操作码和地址后, 在所选定地址的存储器中存储的数据被移出到 SO 线上。继续提供时钟脉冲可接着读出在下一地址的存储器中存储的数据。在每个数据字节移出之后, 地址自动增量到下一个较高的地址。当达到最高地址时, 地址计数器翻转至地址 0000H, 使读周期无限地继续下去。把 $\overline{\text{CS}}$ 置为高电平可以终止操作, 时序见图 4-33。

图 4-33 读 E²PROM 阵列时序

为了读状态寄存器, 首先要把 $\overline{\text{CS}}$ 拉至低电平以选择芯片, 然后是 8 位 RDSR 指令。在发出读状态寄存器操作之后, 状态寄存器的内容被移出至 SO 线上, 时序如图 4-34 所示。

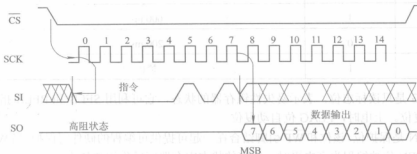


图 4-34 读状态寄存器时序

(5) 写数据。在把数据写入器件之前, 必须首先发出 WREN 指令, 将“写使能”锁存

器置位, 见图 4-35。 $\overline{\text{CS}}$ 首先被拉至低电平, 然后 WREN 指令由时钟同步送入器件。在指令的全部 8 位被发送后, 必须接着使 $\overline{\text{CS}}$ 变为高电平而继续写操作。

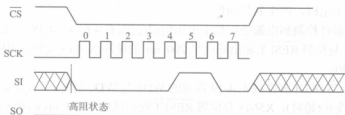


图 4-35 写允许时序

为了把数据写入 E^2PROM 存储器阵列, 用户要发出 WRITE 指令, 接着是 16 位地址, 然后是要写的数。任何不用的地址被规定为“0”。写操作至少要 32 个时钟。在此操作期内, $\overline{\text{CS}}$ 必须变为低电平。如果地址计数器达到页的末尾而时钟仍继续, 那么计数器将“翻转”到页的首址且重写任何已写的数据。

为了完成页写操作(字节或页写), 只能在时钟同步输入最后一个数据字节的位 0 之后把 $\overline{\text{CS}}$ 变为高电平。如果在其他的任何时候使之变为高电平, 则将不能完成写操作, 见图 4-36。

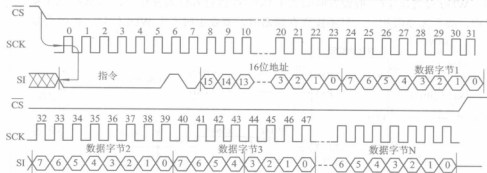


图 4-36 写数据时序

为了写状态寄存器的内容, WRSR 指令要后接将要写入的数据, 见图 4-37。数据位 D0 和 D1 必须是“0”。

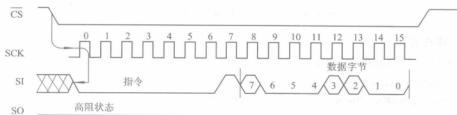


图 4-37 写状态寄存器时序

在写的过程中, 在状态寄存器或 E^2PROM 写时序之后可以读状态寄存器以检查 WIP 位。在此期间, WIP 位将是高电平。

2) 用作 CPU 监视器

当 X5643/45 用作 CPU 监视器时,有以下三种方式可以产生复位动作。

第一种是当上电时,产生复位动作。

第二种是当器件检测到电源电压低于电压门限(门限有 4.65 V、4.38 V、2.94 V 和 2.63 V 几种)时, X5643 复位脚 $\overline{\text{RESET}}$ 输出低电平 200 ms 的脉冲(X5645 复位脚 $\overline{\text{RESET}}$ 输出高电平 200 ms 的脉冲)。

第三种是当 $\overline{\text{CS}}/\text{WDI}$ 脚在按表 4-19 设置的 WDI 与 WD0 的时间内,未接收到“高到低”或“低到高”的变化(超时), X5643 复位脚 $\overline{\text{RESET}}$ 输出低电平 200 ms 的脉冲(X5645 复位脚 $\overline{\text{RESET}}$ 输出高电平 200 ms 的脉冲)。

复位输出是漏极开路输出,使用时要接上拉电阻。

4.5.2 应用电路与编程

1. 应用电路

X5645 器件与 STC89C52 单片机的典型连接电路如图 4-38 所示。当上电时,复位正脉冲持续 200 ms。正常工作时,通过 P1.5 产生“喂狗”脉冲,喂狗的时间间隔由表 4-19 中的 WDI 、 WD0 设置决定(一般设置时间为 1.4 s)。在进行存储器操作时,因 WDI 和 $\overline{\text{CS}}$ 是同一个引脚,故不需要喂狗。一旦不操作存储器时,就要定时产生 WDI 信号,否则会复位。

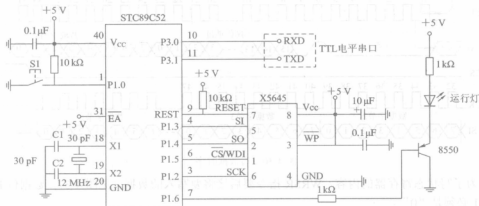


图 4-38 X5645 与 STC89C52 的典型连接

2. 编程方法

用 C51 实现的源程序如下:

```
#include <reg52.h>

#define WREN 0x06 /*写允许指令*/
#define WRDI 0x04 /*写禁止指令*/
#define RFLB 0x04 /*复位使能标志*/
#define RDSR 0x05 /*读状态寄存器指令*/
#define WRSR 0x01 /*写状态寄存器指令*/
```

```

#define READ    0x03        /*读数据指令*/
#define WRITE   0x02        /*写数据指令*/

sbit mdo = P1^4;            /*定义 SPI 输出口*/
sbit mdi = P1^3;            /*定义 SPI 输入口*/
sbit mcs = P1^5;            /*X5645 的片选*/
sbit WDI = P1^5;            /*定义看门狗输入端*/
sbit mscl = P1^2;           /*定义 SPI 时钟口*/
sbit LED = P1^6;            /*定义运行灯输出口*/

unsigned char xdata wrbuffer[128]; /*写缓冲区*/
unsigned char sr=0;          /*全局变量*/
void delay(unsigned char x)    /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

/*发送字节函数，注意这里没有片选，在调用时根据实际情况加入*/
void sendbyte(unsigned char sedata) /*发送 8 位数据函数*/
{
    unsigned char i;
    for(i=0; i<8; i++)
    {
        mdi=(bit)(sedata & 0x80); /*先发送高位*/
        sedata<<=1;
        mscl=1; delay(2);
        mscl=0; delay(2);
    }
}

unsigned char receivebyte(void) /*接收 8 位数据函数*/
{
    unsigned char redata=0, i;
    for(i=0; i<8; i++)
    {
        redata<<=1;
        if(mdo==1) redata++;
        mscl=1; delay(2);
        mscl=0; delay(2);
    }
    return(redata); /*返回结果*/
}

/*适用于写允许、写禁止的操作*/
void send_cmd(unsigned char secmd) /*单字节命令*/
{

```

```

delay(5);
sendbyte(secmd);          /*发送命令*/
delay(5);
}

void send_cmd_X5645(unsigned char secmd) /*操作 X5645 单字节命令*/
{
    mcs=0;                /*使 X5645 的 cs=0*/
    send_cmd(secmd);      /*发送指令*/
    mcs=1;                /*使 X5645 的 cs=1*/
}

unsigned char rdsr_X5645(void) /*读状态寄存器命令*/
{
    unsigned char ch=0;
    mcs=0;                /*使 X5645 的 cs=0*/
    delay(10);            /*延时*/
    sendbyte(RDSR);        /*发送命令(RDSR)0x05*/
    ch=receivebyte();      /*读数据, 值在 ch 中*/
    delay(5);             /*延时*/
    mcs=1;                /*使 X5645 的 cs=1*/
    return ch;            /*返回数据*/
}

void W_WIP(void) /*查看 WIP 是否为 0*/
{
    while( (rdsr_X5645() & 0x01)==0x01);
}

void wrsr_X5645(unsigned char sesr) /*写状态寄存器命令*/
{
    mcs=0;                /*使 X5645 的 cs=0*/
    delay(10);            /*延时*/
    sendbyte(WRSR);        /*写状态寄存器命令 0x01*/
    sendbyte(sesr);        /*要写的内容*/
    delay(5);             /*延时 */
    mcs=1;                /*使 X5645 的 cs=1*/
}

void flashInit(void) /*X5645 初始化*/
{
    mcs=1; msc1=0;
}

void read_X5645(unsigned int address, unsigned int numb) /*可以读任意地址的数据*/
{

```

```

unsigned int i;
unsigned char abh, abl;
abh=address/256;          /*取出高 8 位地址*/
abl=address%256;          /*取出低 8 位地址*/
mcs=0;                    /*使 X5645 的 cs=0*/
delay(10);
sendbyte(READ);           /*读数据命令*/
sendbyte(abh);             /*送高 8 位地址*/
sendbyte(abl);             /*送低 8 位地址*/
for(j=0; j<numb; j++)
    wrbuffer[j]=receivebyte(); /*接收 8 位数据函数*/
delay(5);
mcs=1;                    /*使 X5645 的 cs=1*/
}

/*给 X5645 任意地址写入任意个数据*/
void write_X5645(unsigned int address, unsigned int numb) /*写任意数据函数*/
{
    unsigned int i;
    unsigned char abh, abl;
    abh=address/256;       /*取出高 8 位地址*/
    abl=address%256;       /*取出低 8 位地址*/
    send_cmd_X5645(WREN); /*写允许*/
    delay(5);
    mcs=0;
    delay(10);
    sendbyte(WRITE);       /*写指令*/
    sendbyte(abh);          /*高 8 位地址*/
    sendbyte(abl);          /*低 8 位地址*/
    for(j=0; j<numb; j++)
        sendbyte(wrbuffer[j]); /*写入 numb 个数据*/
    delay(5);
    mcs=1;
}

unsigned char xdata xdataA[128]; /*定义的数据区*/
void mAin(void)
{
    unsigned int i;
    wrsr_X5645(0x00);          /*设置看门狗时间为 1.4 s, 存储器全部开放(可写)*/
    for(i=0; i<128; i++)
    {

```

```

        xdataA[i]=i;          /*产生的模拟数据*/
        WDI= ~WDI;          /*喂狗*/
    }

    flashInit();             /*初始化 flash*/

    for(i=0; i<128; i++)
        wrbuffer[i]=xdataA[i]; /*向写入缓冲器送数*/

    write_X5645(0x200, 20);    /*向 X5645 的 200H~200H+20 连续送数据*/
    W_WIP();                  /*查询 WIP 是否为 0*/
    read_X5645(0x200, 20);     /*从 X5645 中连续读出 20 个数据*/

    while(1)
    {
        WDI= ~WDI;          /*喂狗*/
        LED= ~LED;          /*运行灯闪烁*/
        delay(90);
    }
}

```

4.6 X84161/641/129 低成本 E²PROM 存储器

4.6.1 硬件与功能描述

X84161/641/129 是低成本串行接口的 E²PROM 存储器, 不需要专门的硬件接口, 很容易和微控制器总线连接, 可方便地与 8 位、16 位、32 位和 64 位 CPU 系统通信。其传输速率高达 10 MHz, 访问时间小于 25 ns, 同大多数主机的访问无需“等待”操作。该器件具有字节和页写能力, 可通过写保护引脚防止对存储器数据的干扰破坏。

1. 主要性能特点

- (1) 典型非易失性写周期时间为 2 ms(典型值);
- (2) 编程次数: >100 000 次;
- (3) 数据保存时间: >100 年;
- (4) 器件工作电压: 1.8~5.5 V;
- (5) 正常工作电流: <1 mA;
- (6) 等待电流: <1 μ A;
- (7) 接口传输速率: 10 MHz(典型值);
- (8) 具有字节和页(32 字节)写能力;
- (9) 工作温度: -40~+85℃(工业级), -55~+125℃(军品级)。

2. 内部结构与引脚说明

X84161/641/129 的内部结构与引脚排列如图 4-39 所示。

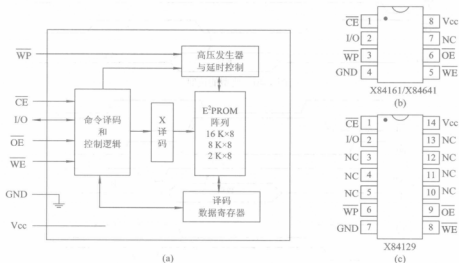


图 4-39 X84161/641/129 的内部结构与引脚排列

(a) 内部结构；(b)、(c) 引脚排列

X84161/641/129 系列器件的引脚排列常用的有 SOP-8 与 SOP-14 两种。其中：

(1) $\overline{\text{CE}}$ 是器件片选使能端，低电平有效。当 $\overline{\text{CE}}$ 为高电平时，芯片不被选中，I/O 引脚处于高阻抗状态，除非非易失性写操作正在进行，器件将处于等待状态。

(2) I/O 是数据输入/输出端。

(3) $\overline{\text{WP}}$ 是写保护端，低电平有效。当写保护输入为低电平时，对器件的非易失性写被禁止；当 $\overline{\text{WP}}$ 为高电平时，所有功能包括非易失性写均正常工作。如果非易失性写周期正在进行，则 $\overline{\text{WP}}$ 变为低电平将对已在进行的周期没有影响，但是将禁止任何额外的非易失性写周期。

(4) $\overline{\text{WE}}$ 是写使能端。为了把数据或命令序列写入器件，写使能输入必须为低电平。

(5) $\overline{\text{OE}}$ 是输出使能端。为了允许输出缓冲器工作，并在 I/O 线上从器件读出数据，输出使能端必须为低电平。

(6) NC 是空脚。

(7) Vcc 是电源输入端。

(8) GND 是参考地。

3. 使用说明

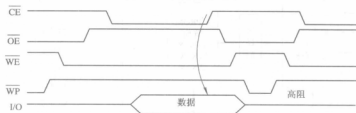
X84161/641/129 可与绝大部分微处理器总线接口。标准的 $\overline{\text{CE}}$ 、 $\overline{\text{OE}}$ 和 $\overline{\text{WE}}$ 信号控制读和写操作，而单根 I/O 线用于串行数据(或命令)的收/发。图 4-40 是典型的读时序。

X84161/641/129 的写模式有两种方法。第一种是在 $\overline{\text{CE}}$ 控制下的写数据模式，即在 $\overline{\text{CE}}$ 的下降沿将数据送到 I/O 口上，在 $\overline{\text{CE}}$ 结束的上升沿把数据锁入内部寄存器，时序如图 4-41(a)所示。第二种是在 $\overline{\text{WE}}$ 控制下的写数据模式，即在 $\overline{\text{WE}}$ 的下降沿将数据送到 I/O 口上，在 $\overline{\text{WE}}$

结束的上升沿把数据锁入内部寄存器, 时序如图 4-41(b)所示。



图 4-40 典型的读时序



(a)



(b)

图 4-41 写数据时序

(a) \overline{CE} 控制的写时序; (b) \overline{WE} 控制的写时序

1) 数据定时操作

I/O 线上输入的数据由 \overline{WE} 或 \overline{CE} 上升沿输入锁存。I/O 线上输出的数据在 \overline{OE} 和 \overline{CE} 二者同时为低电平时有效输出。要注意确保在 \overline{CE} 为低电平时不能使 \overline{WE} 和 \overline{OE} 二者同时为低电平。

2) 读时序操作

读时序包括发送 16 位地址, 后随串行的数据读出。通过向器件发出 16 个单独的写周期, 在写周期之间没有读周期来写入地址。在 I/O 线上地址被串行发送, 最高有效位在前。注意, 此序列是完全静态的, 没有特殊的定时限制, 当器件 \overline{CE} 引脚为高电平时, 处理器能自由地在总线上完成其他任务。一旦发送了 16 个地址位, 便可发出 8 个单独的读周期在 I/O 线上读出数据字节。

3) 顺序读操作

在每一个数据字节被读出之后, 字节地址将自动增量至下一较高地址。通过继续发出读周期, 存储在下一地址存储器中的数据可被顺序读出。当达到阵列的最高地址时, 地址计数器翻转到地址 0000H, 读操作可无限地继续下去。

4) 复位时序

复位时序可在任何时候通过执行读/写“0”进行(见图 4-42 所示的读时序)。这就打断了多次读或写周期序列, 该序列通常用于从器件读出或写入器件的操作。复位时序可在任何

时刻使用以中断或终止顺序读或页操作。一旦写“0”周期完成,器件便复位(除非非易失性写周期正在进行)。此序列中的第二个读周期以及任何进一步的读周期将在 I/O 引脚上读出高电平,直至发出有效的读时序为止。在读和写序列的开始必须发出复位序列以确保器件适当地启动这些操作。

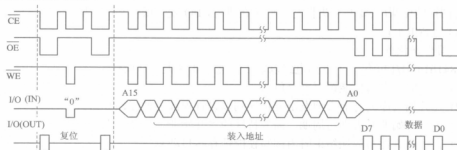


图 4-42 读时序

5) 写时序

非易失性写时序包括发送复位序列、16 位地址、多达 32 字节的数据以及接着的“启动非易失性写周期”命令序列,如图 4-43 所示。

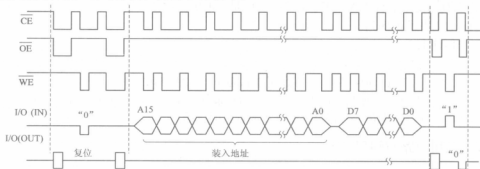


图 4-43 写时序

首先发出复位序列以便设置写使能锁存器。通过向器件发出 16 个单独的写周期(WE 和 CE 为低电平, OE 为高电平)且在写周期之间没有任何读周期,串行写入地址。在 I/O 引脚上串行发送地址,最高位在前。通过发出 8 位数的写周期写入多达 32 个字节的数据。同样,在写周期之间不允许有读周期。

通过发出专门的读/写“1”,读序列启动非易失性写周期。第一个读周期结束页装载,然后写“1”继续开始非易失性写周期。器件识别 32 字节(页)。

在向器件发送数据时,企图超出页的地址上限将导致地址计数器“回绕”至页的首地址,在那里可继续数据的装载。因此,发送多于 256 个连续的数据位将导致重写先前的数据。

如果发出部分或不完整的写序列,那么将不能开始非易失性写周期。在非易失性写周期完成以及防止偶然写操作的无效写之后,内部写使能锁存器被复位。注意,此序列是完全静态的,没有特殊的定时限制,当器件芯片使能引脚(CE)为高电平时,处理器能自由地在总线上完成其他任务。

6) 非易失性写状态

在任何时候通过简单读取器件上 I/O 引脚的状态可以决定非易失性写周期的状态。在 $\overline{\text{OE}}$ 和 $\overline{\text{CE}}$ 为低电平且 $\overline{\text{WE}}$ 为高电平时读此引脚。在非易失性写周期内, I/O 引脚为低电平。当非易失性写周期完成时, I/O 引脚变为高电平。在非易失性写周期内, 也可发出复位序列并具有同样的结果(只要非易失性写周期正在进行, I/O 就为低电平; 当非易失性写周期完成时, I/O 为高电平)。

7) 写保护操作

X84161/641/129 包含了下列电路以防止偶然的非易失性写操作。

(1) 上电时内部写使能锁存器复位。

(2) 在开始写序列之前必须发出复位序列以设置内部写使能锁存器。

(3) 为了开始非易失性写周期, 需要特殊的“开始非易失性写”命令序列。

(4) 在非易失性写周期结束时内部写使能锁存器自动复位。

(5) 只要 $\overline{\text{WP}}$ 引脚为低电平, 内部写使能锁存器就被复位并保持复位状态。 $\overline{\text{WP}}$ 引脚为低电平将中断所有非易失性写周期。

(6) 无效写操作时, 内部写使能锁存器复位。

8) 低功耗运用

在下列情况下器件进入空闲状态, 并吸收最小的电流。

(1) 进入非法的序列。

(2) 器件上电。

(3) 非易失性写操作完成。

在顺序读正在进行时, 器件保持在工作状态下。此状态比空闲状态吸收较大的电流, 但不像读期间本身吸收那么多的电流。为了回到最低功率的状态, 可通过在读操作的最后一位之后写“1”来建立一个无效状态, 实现省电的目的。

4.6.2 应用电路与编程

1. 应用电路

X84161/641 系列器件与 STC89C54 单片机的连接如图 4-44 所示。

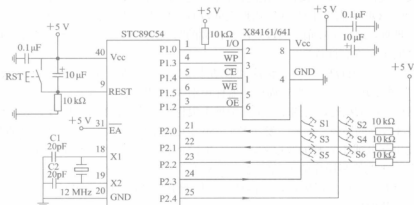


图 4-44 X84161/641 与 STC89C54 的应用电路

2. 编程方法

根据 X84161/641/129 的工作时序和图 4-44 所示的硬件连接方式, 用 C51 所实现的相关函数如下:

```
#include <reg52.h>

sbit io=P1^0;          /*定义 I/O 输入/输出口*/
sbit wp=P1^3;          /*定义保护口*/
sbit ce=P1^4;          /*X84161 的片选*/
sbit we=P1^5;          /*X84161 的写允许*/
sbit oe=P1^2;          /*定义读允许引脚*/
sbit key_i1=P2^0;      /*定义按键输入 1*/
sbit key_i2=P2^1;      /*定义按键输入 2*/
sbit key_i3=P2^2;      /*定义按键输入 3*/
sbit key_o1=P2^3;      /*定义按键输出 1*/
sbit key_o2=P2^4;      /*定义按键输出 2*/

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

void reset(void)          /*复位函数*/
{
    ce=1; oe=1; we=1; io=1;
    ce=0; oe=0; delay(5);
    oe=1; ce=1; delay(3);
    ce=0; we=0; io=0; delay(3);
    io=1; we=1; delay(4);
    ce=1; delay(4);
    ce=0; oe=0; delay(4);
    oe=1; ce=1;
}

void reset_1(void)        /*发出 io=1 函数*/
{
    ce=1; oe=1; we=1; io=1;
    ce=0; oe=0; io=0; delay(5);
    oe=1; ce=1; delay(8);
    ce=0; we=0; io=1; delay(5);
    io=0; we=1; ce=1; delay(5);
    ce=0; oe=0; delay(3);
    oe=1; ce=1;
```

```

    }
    /*发送字节函数, 注意 oe=1, 在调用时根据实际情况加入*/
    void sendbyte(unsigned char sedata)    /*发送 8 位数据函数*/
    {
        unsigned char i;
        ce=1; we=1;
        for(i=0; i<8; i++)
        {
            io=(bit)(sedata & 0x80);    /*先发送高位*/
            sedata<<=1;
            ce=0; we=0; delay(2);
            we=1; ce=1; delay(2);
        }
    }

    unsigned char receivebyte()    /*接收 8 位数据函数*/
    {
        unsigned char redata=0,i;
        ce=1; oe=1; we=1;
        for(i=0; i<8; i++)
        {
            redata<<=1;
            ce=0; oe=0; delay(2);
            oe=1; ce=1;
            if(io==1) redata++;
            delay(2);
        }
        return(redata);    /*返回结果*/
    }

    void send_AB_X84161(unsigned int address) /*发送地址*/
    {
        unsigned char abh,abl;
        abh=address/256;
        abl=address%256;
        reset(); /*复位*/
        oe=1;
        sendbyte(abh);    /*发送高位地址*/
        sendbyte(abl);    /*发送低位地址*/
    }

    void flashInit(void) /*X5645 初始化*/

```

```

    {
        ce=1; oe=1; we=1; wp=1;
    }

unsigned char read_X84161(unsigned int address) /*可以读任意地址的数据*/
{
    unsigned char x;
    send_AB_X84161(address); /*送地址*/
    we=1;
    x=receivebyte(); /*读一个8位数据*/
    return(x); /*返回数据*/
}

/*给 X84161 的任意地址写入一个任意数据*/
void write_X84161(unsigned int address,unsigned char numb) /*写任意数据函数*/
{
    oe=1;
    send_AB_X84161(address); /*送地址*/
    sendbyte(numb); /*写入一个8位数据*/
    reset_1();
}

void main(void) /*主函数*/
{
    unsigned char i;
    unsigned char y[32];
    flashInit(); /*初始化 flash*/
    for(i=0; i<32; i++)
        write_X84161(i,0x55); /*向 X84161 写入 0x55 的数据*/
    delay(0x55);
    for(i=0; i,32; i++)
        y[i]=read_X84161(i); /*从 X84161 连续读出 32 个数据*/
    while(1); /*结束*/
}

```

4.7 93AA46/56/66 低压低功耗 E²PROM 存储器

4.7.1 硬件与功能描述

93AA46/56/66 是 1k/2k/4k 低压串行电可擦除 PROM 存储器。器件通过 ORG 引脚可配置成 $\times 8$ 位或 $\times 16$ 位的存储组织结构(93AA46 器件为 128×8 或 64×16 ; 93AA56 器件为 256×8 或 128×16 ; 93AA66 器件为 512×8 或 256×16)。Microchip 公司采用先进的 CMOS

技术,使器件可以在极小的功耗下可靠工作。93AA46/56/66 具有低电压、低成本的特点,很适合于电池供电系统,封装采用标准的 DIP-8 或 SOP-8 形式,接口采用标准的 3 线串口方式。

1. 主要性能特点

- (1) 器件工作电压: 1.8~5.5 V;
- (2) 典型的读电流: 在 1.8 V 时仅 70 μ A;
- (3) 等待电流: $<2 \mu$ A;
- (4) 存储组织结构选择: $\times 8$ 或 $\times 16$;
- (5) 典型非易失性写周期时间: 4 ms(典型值);
- (6) 编程次数: >100 万次;
- (7) 数据保存时间: >200 年;
- (8) 接口传输速率: 2 MHz(典型值);
- (9) ESD 引脚保护: 4 kV(典型值);
- (10) 工作温度: $0\sim+70^{\circ}\text{C}$ (民品), $-40\sim+85^{\circ}\text{C}$ (工业)。

2. 内部结构与引脚说明

93AA46/56/66 的内部结构与引脚排列如图 4-45 所示。

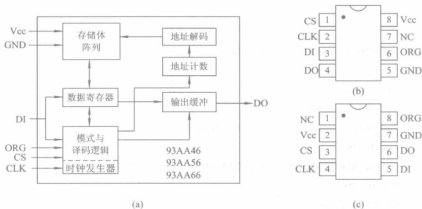


图 4-45 93AA46/56/66 的内部结构与引脚排列

(a) 内部结构; (b)、(c) 引脚排列

93AA46/56/66 存储器的引脚排列有 DIP-8 和 SOP-8 两种, 分别如图 4-45(b)、(c)所示。其中:

- (1) CS 是片选端, 高电平有效。在低电平时不会选中器件, 但迫使器件进入标准模式。
- (2) CLK 是器件的串行时钟输入端, 数据在时钟的上升沿输入或输出。
- (3) DI 是串行数据的输入端。
- (4) DO 是串行数据的输出端。该脚也提供了在擦除或编程时的“Ready/Busy”状态。
- (5) ORG 是存储组织选择端。当 ORG 接 Vcc 时, 是 $\times 16$ 结构; 当 ORG 接地时, 是 $\times 8$ 结构。

(6) Vcc 是电源输入端。

(7) GND 是参考地。

(8) NC 表示不用。

3. 器件的使用

当 ORG 接高电平时, 器件是 $\times 16$ 的数据结构; 当 ORG 接低电平时, 器件是 $\times 8$ 的数据结构。数据(DI)的输入是在时钟的上升沿将数据输入, 在数据输入时, 通常输出端(DO)呈高阻状态(除非在编程或擦除期间 DO 用作“Ready/Busy”信号)。器件在编程或擦除期间, DO 为低电平, 表示“器件正在处理”, 即“Busy”; 当 DO 为高电平时, 表示“器件已经空闲”, 即“Ready”。在 CS 为低电平时, DO 进入高阻状态。

1) 操作指令

93AA46/56/66 的操作指令如表 4-21 与表 4-22 所示。其中, READ 是读指令, EWEN 是写允许指令, ERASE 是擦除指令, REAL 是全部擦除指令, WRITE 是写指令, WRAL 是全部写指令, EWDS 是写禁止指令。

表 4-21 ORG = 1($\times 16$)指令集

指令	代码	地 址	数据进入	数据输出	时钟数	型 号
READ	110	A5 A4 A3 A2 A1 A0	—	D15~D0	25	93AA46
		x A6 A5 A4 A3 A2 A1 A0			27	93AA56
		A7 A6 A5 A4 A3 A2 A1 A0			27	93AA66
EWEN	100	1 1 x x x x	—	高阻	9	93AA46
		1 1 x x x x x x			11	93AA56
		1 1 x x x x x x			11	93AA66
ERASE	111	A5 A4 A3 A2 A1 A0	—	Ready/Busy	9	93AA46
		x A6 A5 A4 A3 A2 A1 A0			11	93AA56
		A7 A6 A5 A4 A3 A2 A1 A0			11	93AA66
REAL	100	1 0 x x x x	—	Ready/Busy	9	93AA46
		1 0 x x x x x x			11	93AA56
		1 0 x x x x x x			11	93AA66
WRITE	101	A5 A4 A3 A2 A1 A0	D15~D0	Ready/Busy	25	93AA46
		x A6 A5 A4 A3 A2 A1 A0			27	93AA56
		A7 A6 A5 A4 A3 A2 A1 A0			27	93AA66
WRAL	100	0 1 x x x x	D15~D0	Ready/Busy	25	93AA46
		0 1 x x x x x x			27	93AA56
		0 1 x x x x x x			27	93AA66
EWDS	100	0 0 x x x x	—	高阻	9	93AA46
		0 0 x x x x x x			11	93AA56
		0 0 x x x x x x			11	93AA66

表 4-22 ORG = 0(×8)指令集

指令	代码	地 址	数据进入	数据输出	时钟数	型 号
READ	110	A6 A5 A4 A3 A2 A1 A0	—	D7~D0	18	93AA46
		x A7 A6 A5 A4 A3 A2 A1 A0			20	93AA56
		A8 A7 A6 A5 A4 A3 A2 A1 A0			20	93AA66
EWEN	100	1 1 x x x x x	—	高阻	10	93AA46
		1 1 x x x x x x x			12	93AA56
		1 1 x x x x x x x			12	93AA66
ERASE	111	A6 A5 A4 A3 A2 A1 A0	—	Ready/ $\overline{\text{Busy}}$	10	93AA46
		x A7 A6 A5 A4 A3 A2 A1 A0			12	93AA56
		A8 A7 A6 A5 A4 A3 A2 A1 A0			12	93AA66
REAL	100	1 0 x x x x x	—	Ready/ $\overline{\text{Busy}}$	10	93AA46
		1 0 x x x x x x x			12	93AA56
		1 0 x x x x x x x			12	93AA66
WRITE	101	A6 A5 A4 A3 A2 A1 A0	D7~D0	Ready/ $\overline{\text{Busy}}$	18	93AA46
		x A7 A6 A5 A4 A3 A2 A1 A0			20	93AA56
		A8 A7 A6 A5 A4 A3 A2 A1 A0			20	93AA66
WRAL	100	0 1 x x x x x	D7~D0	Ready/ $\overline{\text{Busy}}$	18	93AA46
		0 1 x x x x x x x			20	93AA56
		0 1 x x x x x x x			20	93AA66
EWDS	100	0 0 x x x x x	—	高阻	10	93AA46
		0 0 x x x x x x x			12	93AA56
		0 0 x x x x x x x			12	93AA66

2) 操作时序

(1) 读时序(READ 指令)。93AA46/56/66 的读时序如图 4-46 所示。其序列是: “110+地址+输出数据”。这种指令可以连续读数据(内部地址会自动增1)。

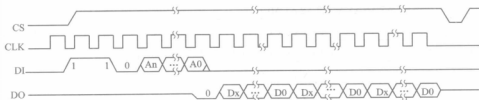


图 4-46 读时序

(2) 写允许(EWEN 指令)。93AA46/56/66 的写允许时序如图 4-47 所示。其序列是：“100+11x...x”。这种指令的使用是在“写数据或擦除数据”前要先操作指令序列。

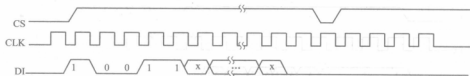


图 4-47 写允许时序

(3) 写禁止(EWDS 指令)。93AA46/56/66 的写禁止时序如图 4-48 所示。其序列是：“100+00x...x”。这种指令是在“写数据”后使用，是要保护所存的数据而进行的操作指令。

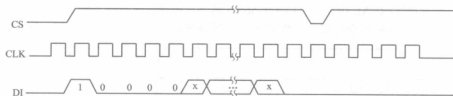


图 4-48 写禁止时序

(4) 写数据(WRITE 指令)。93AA46/56/66 的写数据时序如图 4-49 所示。其序列是：“101+地址+数据”。这种指令可在该地址上写入数据。

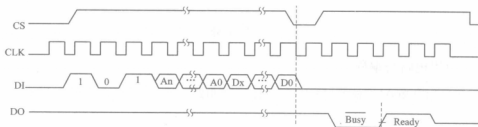


图 4-49 写数据时序

(5) 全写数据(WRAL 指令)。93AA46/56/66 的全写数据时序如图 4-50 所示。其序列是：“100+01x...x+数据”。这种指令可将指定数据全部写入存储器(需要+5 V 供电, 约 16 ms 时间)。

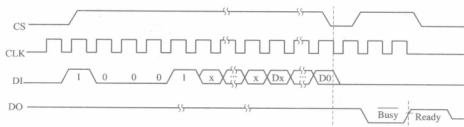


图 4-50 全写数据时序

(6) 擦除数据(ERASE 指令)。93AA46/56/66 的擦除数据时序如图 4-51 所示。其序列是：“111+地址”。这种指令可将指定地址的数据擦除(擦除结果为“1”)。

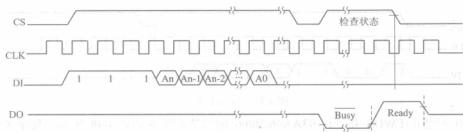


图 4-51 擦除数据时序

(7) 整片擦除(REAL 指令)。93AA46/56/66 的整片擦除时序如图 4-52 所示。其序列是：“100+10x...x”。这种指令可将整片数据全部擦除(全部擦除需要 5 V 供电, 擦除结果为“1”)。

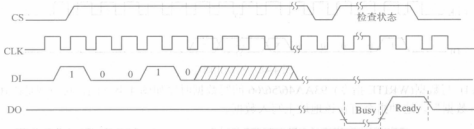


图 4-52 整片擦除时序

4.7.2 应用电路与编程

93AA66 E²PROM 存储器与 8 位单片机的典型电路如图 4-53 所示。

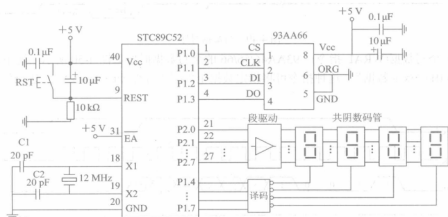


图 4-53 93AA66 与单片机的典型电路

编程时,只要按照表 4-22 与图 4-46~图 4-52 的时序就可方便地实现数据的存储。由 C51 构建的源程序如下:

```
#include <reg52.h>

sbit cs = P1^0;          /*定义片选端*/
sbit clk = P1^1;          /*定义时钟引脚*/
sbit di = P1^2;          /*定义数据的输入脚*/
sbit sdo = P1^3;          /*定义数据的输出脚*/

void delay(unsigned char x) /*延时函数*/
{
    unsigned char i;
    for(i=0; i<x; i++);
}

void flashInit(void) /*93Axx 初始化*/
{
    cs=0; clk=0; di=0;
}

/*sedata 是要发送的命令数据, type 是器件型号*/
/*适合整片擦除、写禁止、写允许指令*/
void sendbyte(unsigned int sedata,unsigned char type) /*发 x 位数据函数*/
{
    unsigned char i,n;
    unsigned int dat;
    cs=0; clk=0; di=0; sdo=1;
    dat=sedata;
    if(type==(0x46) n=10;
    else n=12;
    clk=1; delay(2);
    clk=0; delay(2); cs=1;
    for(i=0; i<n; i++)
    {
        di=(bit)(dat & 0x8000); /*先发送高位*/
        dat<<=1;
        clk=1; delay(2);
        clk=0; delay(2);
    }
    cs=0;
}

/*type 是器件型号, address 是地址*/
void erase_byte(address,type) /*擦除指定地址的内容*/
{
    unsigned int address;
```

```

unsigned char type;

{
    unsigned char i,n;
    unsigned int dat;
    cs=0; clk=0; di=0; sdo=1;
    if(type==0x46)
        {dat=address<<6; dat= dat |0xe000; n=10; }
    else {dat=address<<4; dat= dat |0xe000; n=12; }
    clk=1; delay(2);
    clk=0; delay(2); cs=1;
    for(i=0; i<n; i++)
    {
        di=(bit)(dat & 0x8000);    /*先发送高位*/
        dat<<=1;
        clk=1; delay(3);
        clk=0; delay(2);
    }
    cs=0;
    while(sdo==0);                /*判断是否忙*/
}

/*type 是器件类型(46/56/66), bdat 是数据, address 是地址*/
/*适合 3 种器件在指定地址上发送数据*/
void w_data(address, type, bdat)    /*写 8 位数据函数*/
{
    unsigned int address;
    unsigned char type, bdat;
    { unsigned char i,j;
        unsigned int dat;
        cs=0; clk=0; di=0; sdo=1;
        if(type==0x46) {dat=address<<6; dat= dat |0xa000; j=10; }
        else {dat=address<<4; dat= dat |0xa000; j=12; }
        clk=1; delay(2);
        clk=0; delay(2); cs=1;
        for(i=0; i<j; i++)
        {
            di=(bit)(dat & 0x8000);
            dat<<=1;
            clk=1; delay(2);
            clk=0; delay(2);
        }
    }
}

```

```

        for(i=0; i<8; i++)      /*发送 8 位数据*/
        {
            di=(bit)(bdat & 0x80);
            bdat<<=1;
            clk=1; delay(2);
            clk=0; delay(2);
        }
        cs=0;
    }

void weat_do(void)              /*判断是否忙*/
{
    while(sdo==0);
}

/*从给定地址读出数据, 接收 8 位数据函数*/
unsigned char receivebyte(unsigned char type,unsigned int address)
{
    unsigned char i,j,bdat=0;
    unsigned int dat;
    if(type==0x46) {dat=address<<6; dat= dat | 0xc000; j=10; }
    else {dat=address<<4; dat=dat | 0xc000; j=12; }
    flashInit();                /*初始化*/
    clk=1; delay(3);
    clk=0; delay(3); cs=1;
    for(i=0; i<j; i++)
    {
        di=(bit)(dat & 0x8000);
        dat<<=1;
        clk=1; delay(2);
        clk=0; delay(2);
    }
    sdo=1;
    for(i=0; i<8; i++)
    {
        bdat<<=1;
        clk=1; delay(2);
        if(sdo==1) bdat++;
        clk=0;
    }
    cs=0;
}

```

```

        return(bdat);           /*返回结果*/
    }

void main(void)                /*主函数部分*/
{
    unsigned int i;
    unsigned char y[32];
    flashInit();               /*初始化 flash*/
    sendbyte(0x90,0x66);       /*擦除整个芯片内容*/
    sendbyte(0x98,0x66);       /*写允许*/
    for(i=0; i<32; i++)
    {
        w_data(i, 0x66, i);     /*给地址 i 写入数据 i*/
        weat_do();              /*判断是否写完*/
    }
    sendbyte(0x80,0x66);       /*写禁止*/
    for(i=0; i,32; i++)
        y[i]=receivebyte(0x66,i); /*从 i 地址读数*/
    while(1);                  /*结束等待*/
}

```

第5章 通信模块的使用与编程

5.1 GTM900 GSM/GPRS 无线通信模块

5.1.1 硬件与功能描述

GTM900 无线模块是一款三频段 GSM/GPRS 的无线通信模块,它支持标准的 AT 命令及增强的 AT 命令,提供丰富的语音和数据通信等多种功能,支持 800 MHz、900 MHz 和 1800 MHz,三频自动选择,通过内部硬件的选择可支持 850 MHz 频段和 1900 MHz 频段。

GTM900 无线模块与 TC35i 等通信模块的功能基本兼容。

1. 主要性能特点

- (1) 提供标准 UART 通信接口,串口速率最大支持 115.2 kb/s;
- (2) 支持分组数据业务和 GPRS CLASS 10 协议;
- (3) 支持 FR EFR HR 和 AMR 的语音编码;
- (4) 支持标准 AT 命令集和扩展 AT 命令集;
- (5) 提供 UART Audio SIM 卡和 Power Control ADC 接口;
- (6) 支持高质量语音业务;
- (7) 支持无线数据业务、电路型数据业务和分组域数据业务;
- (8) 支持短消息(MT、MO)功能;
- (9) 支持来电显示、呼叫转移、呼叫前转、呼叫保持、呼叫等待和三方通话等业务功能;
- (10) 支持组呼、点对点通信和私密呼叫等功能;
- (11) 支持 TCP/IP 和 UDP/IP 协议;
- (12) 模块电源: +3.3~+4.8 V(典型值为+4.0 V);
- (13) 信号电平: +1.85~+3.14 V;
- (14) 空闲状态电流: 4 mA(典型值);
- (15) 通话状态电流: 250 mA(典型值)。

2. 内部结构与引脚说明

GTM900 无线模块的内部组成框图如图 5-1 所示。

GTM900 使用 AT 命令集,通过 UART 接口与外部 CPU 通信,主要实现无线发送和接收、基带处理、音频处理等功能。有了这种简单的串行接口,用户可实现各种通信链路和具有键盘、LCD 显示、语音等各种终端设备。

GTM900 无线模块的接口是一个 40 脚的 FPC 连接器,引脚间距为 0.5 mm,线距为 0.5 mm,

形状为单排弯式表贴带电缆锁紧结构，引线如图 5-2 所示，接口信息含义见表 5-1。

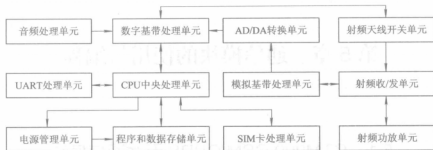


图 5-1 GTM900 无线模块的内部组成框图

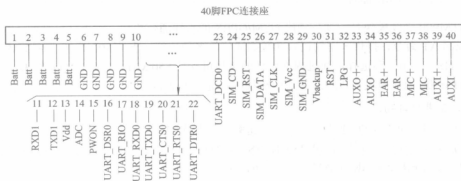


图 5-2 40 脚信号排列

表 5-1 40 脚信息含义

脚号	信息名称	I/O	接口电平	功能描述	备注
1	Batt	I	4.0 V	模块供电电源	最大 4.5 V
2	Batt	I	4.0 V	模块供电电源	最大 4.5 V
3	Batt	I	4.0 V	模块供电电源	最大 4.5 V
4	Batt	I	4.0 V	模块供电电源	最大 4.5 V
5	Batt	I	4.0 V	模块供电电源	最大 4.5 V
6	GND	I/O	—	工作参考地	—
7	GND	I/O	—	工作参考地	—
8	GND	I/O	—	工作参考地	—
9	GND	I/O	—	工作参考地	—
10	GND	I/O	—	工作参考地	—
11	红外串口的 RXD1	O	3.0 V	串口的接收数据，用于调试	—
12	红外串口的 TXD1	I	3.0 V	串口的接收数据，用于调试	—
13	Vdd	O	3.0 V	输出模块正常启动指示信号	“1”有效
14	ADC	I	—	输入模拟数字采样信号(10 位)	尚不支持

续表

脚号	信息名称	I/O	接口电平	功能描述	备注
15	PWON 信号	I	3.0 V	输入开机控制信号,低电平有效	—
16	UART_DSR0	O	3.0 V	标识输出模块串口是否准备好	—
17	UART_RIO	O	3.0 V	输出串口的振铃指示信号	尚不支持
18	UART_RXD0	O	3.0 V	计算机的串口接收信号	—
19	UART_TXD0	I	3.0 V	计算机的串口发送信号	—
20	UART_CTS0	O	3.0 V	计算机的串口接收请求信号	—
21	UART_RTS0	I	3.0 V	计算机的串口发送信号	—
22	UART_DTR0	I	3.0 V	计算机串口是否准备好的信号	—
23	UART_DCD0	O	3.0 V	输出载波检测信号	—
24	SIM_CD	I	3.0 V	输入 SIM 卡是否在位的信号	尚不支持
25	SIM_RST	O	3.0 V	输出 SIM 卡复位信号	—
26	SIM_DATA	I/O	3.0 V	SIM 卡数据输入/输出接口	—
27	SIM_CLK	O	3.0 V	输出 SIM 卡时钟信号	—
28	SIM_VCC	O	3.0 V	输出给 SIM 卡供电的信号	—
29	SIM_GND	I/O	0 V	SIM 卡的接地信号	—
30	Vbackup	I	3.0 V	模块的备份电池	—
31	RST	I	—	复位信号	—
32	LPG	O	—	输出指示灯状态控制信号	—
33	AUXO+	O	—	第二路音频输出信号	—
34	AUXO-	O	—	第二路音频输出信号	—
35	EAR+	O	—	第一路音频输出信号	—
36	EAR-	O	—	第一路音频输出信号	—
37	MIC+	I	—	第一路音频输入信号	—
38	MIC-	I	—	第一路音频输入信号	—
39	AUXI+	I	—	第二路音频输入信号	—
40	AUXI-	I	—	第二路音频输入信号	—

GTM900 提供的天线接口为 GSC 焊盘形式, 外接天线需要通过焊接的方式实现连接, 用户也可以焊接上 GSC 射频连接头, 外接天线通过电缆连接到该连接器上。

3. 串行接口

UART 接口与外界进行串行通信, 支持 3.0 V 电平输入和输出, UART 接口的信号除了 RXD0 和 TXD0 是高电平有效外, 其余信号均为低电平有效。该接口支持可编程的数据宽度、可编程的数据停止位、可编程的奇/偶校验或者没有校验。UART 接口工作的最大速率为 115.2 kb/s, 默认支持 9600 b/s 的速率。

UART 串行接口用到的信号有: UART_DCD0(23 脚, 载波检测)、UART_RIO(17 脚, 振铃指示)、UART_RTS0(21 脚, 请求发送)、UART_TXD0(19 脚, 发送数据)、UART_DSR0(16 脚, 数据设备就绪)、UART_DTR0(22 脚, 数据终端就绪)、UART_CTS0(20 脚, 清除发送)和 UART_RXD0(18 脚, 接收数据)8 个信号。除 UART_TXD0、UART_RXD0 信号之外, 其

余信号都可以接成固定电平。

4. SIM 卡接口

GTM900 可外接 3.0 V 的 SIM 卡, SIM 卡接口信号有: SIM_CLK(27 脚, SIM 卡时钟信号)、SIM_RST (25 脚, SIM 卡复位信号)、SIM_DATA(26 脚, SIM 卡数据线)、SIM_Vcc(28 脚, SIM 卡电源)、SIM_CD(24 脚, SIM 卡在位)和 SIM_GND(29 脚, SIM 卡的地)共 6 个信号。

5. RTC Backup 接口

RTC Backup 电路的主要作用是在模块外部的电源 VBAT 断开或掉电的情况下,能保持模块的实时时钟(Real Time Clock)。例如, GTM900 模块没有 VBAT 电源供电, 此时如果外加“RTC Backup 电路”, 则 GTM900 模块就能保持自动更新时间, 直到外部的“RTC Backup 电源”耗尽; 否则, GTM900 模块会在没有 VBAT 电源供电的情况下, 停止更新时间信息。外接“RTC Backup 电源”的接口电压是 3.0 V, 电流能力为 10 μ A。GTM900 模块能自动地对外部的“RTC Backup 电源”进行充电。接入“RTC Backup 电源”的方法是在信号 Vbackup(30 脚)上加入+3 V 的可充电电池。

6. Audio 接口

GTM900 提供两路音频输入/输出信号, 两路信号均为差分信号。第一路信号有: EAR+(35 脚, 音频输出信号+端)、EAR-(36 脚, 音频输出信号-端)、MIC+(37 脚, 音频输入信号+端)和 MIC-(38 脚, 音频输入信号-端)。第二路信号有: AUXO+(33 脚, 路音频输出信号+端)、AUXO-(34 脚, 音频输出信号-端)、AUXI+(39 脚, 音频输入信号+端)和 AUXI-(40 脚, 音频输入信号-端)。

其中, 第一路音频输入/输出通道的性能更加良好, 配置更加灵活和方便。因此, 如果仅使用一路音频通道, 则此音频通道优选。听筒方式连接时, 无需外加音频放大器。如果同时需要两路音频通路, 例如在固定台上, 则通常推荐如下连接方式: 第一路音频通道用作听筒通道, 第二路音频通道用作免提通道。这时, 第二路的音频通道需要外加音频放大器进行放大, 靠近音频放大器的前后需进行射频滤波。两路音频输入/输出信号的连接方式分别如图 5-3(a)与图 5-3(b)所示。

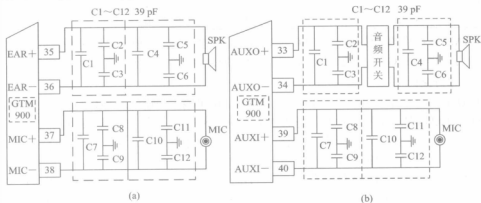


图 5-3 音频电路

(a) 第一路音频电路; (b) 第二路音频电路

由于四路音频信号均是差分对信号,因此需要平行等距离进行 PCB 走线,走线长度尽量最短。两边滤波电路尽量对称,两差分信号间尽量靠近外加包地处理。输出音频差分对信号与输入音频差分对信号通过接地的方式有效隔开,同时需要远离电源、射频和天线等电路。

GTM900 模块在应用 PCB 板设计时,应充分注意各功能模块的区分和隔离。首先电源部分需要提供足够的电流和尽量减少电源的纹波。因为在通过模块进行 GSM 语音通信和免提通话的过程中,需要消耗较大的电流,所以如果 DC/DC 转换的电流供应不足或电压供应不当,则 GSM 模块在时域内的帧切换和 VBAT 上将会产生较大的纹波,其频率主要集中在 217 Hz 上,从而形成音频的“嗞嗞”声。优化和消除的办法是选择性能和电流供给能力较好的电源。在进行 PCB 布局时,电源线尽量靠近 GTM900 模块的 1~5 脚,走线宽度至少 80 mil,可适当增加其宽度, GTM900 模块的 5~10 脚作为回路,地信号走线宽度同样至少 80 mil。

在电源线上需要适当放置滤波和储能电容,同时注意电源部分不能靠近射频信号和天线,尤其是滤波和储能电容的位置需要尽量远离射频信号和天线。第二音频部分的走线和布局需要优先考虑,例如在固定台的应用中,通常会涉及上述四路差分音频弱信号(无论放大前还是放大后),其在 PCB 上的走线需要远离电源、射频信号和天线。走线时差分对信号需要尽量靠近,走线宽度和长度保持一致,并进行包地处理。差分音频信号对之间需要进行隔地处理,隔离间隔大于 25 mil,音频通路的输出信号与音频通路的输入信号也需要隔地处理,隔离间隔大于 25 mil,如果外加滤波电容或磁(电感),则需要注意电路和位置的对称性。

图 5-3 中, 39 pF 的电容主要用于滤除射频信号的干扰。如果 PCB 布局空间允许,则可以同时加上 10 pF 的电容,滤波效果会更好。

MIC+、MIC-、AUXI+和 AUXI-内部已经进行了直流偏置和隔直处理,因此在 GTM900 模块应用电路中,不需要外部再添加麦克风的直流偏置电压和隔直处理。由于考虑到射频信号对音频信号的干扰,因此在 GTM900 模块内部添加有很多滤波电容,同样在 GTM900 模块外部增加的滤波电容(C1~C12)也是必不可少的。

7. LPG 接口

LPG 接口输出不同的信号,可给指示灯指示模块的不同工作状态。其外接指示电路如图 5-4 所示。

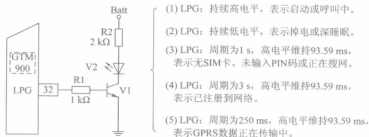


图 5-4 LPG 状态指令电路

8. 电源接口

GTM900 模块的电源标准值应为 4.0 V/2 A, 最小+3.3 V, 最大+4.8 V, 波纹要尽可能小。

5.1.2 工作流程与系统组成

1. 开机流程

当提供给 GTM900 的电源大于 3.3 V, PWON 信号为低电平(维持 10~100 ms)时, GTM900 开始工作。一般地, 在开机寻找网络期间, 用电较大。上电过程如图 5-5(a)所示。

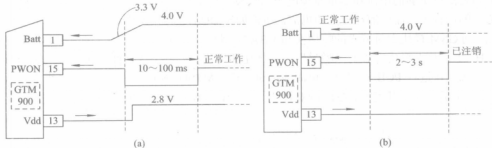


图 5-5 开机与注销流程

(a) 开机; (b) 注销

2. 关机流程

关机流程分为正常关机和紧急关机, 正常关机是由外部 CPU 将 GTM900 的 PWON 信号拉低 2~3 s 使其进入正常关机流程, 保存信息并完成网络注销, 如图 5-5(b)所示。紧急关机是由外部 CPU 直接发送 AT 命令给 GTM900 使其直接下电关机。

3. 系统组成

GTM900 通信模块外接 SIM 卡、UART、CPU、语音、按键、LCD 和电源管理等, 组成的系统框图如图 5-6 所示。如果只用 GTM900 作为无线传输数据(通过 TCP/IP、UDP/IP 协议或短信), 则可省去语音、键盘和 LCD 等部分。

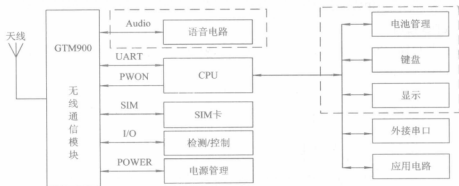


图 5-6 系统组成框图

5.1.3 AT 命令集

操作 GTM900 通信模块, 在 UART 接口的支持下, 通过软件用 AT 命令就可实现所有

的操作。一般来讲, AT 命令包括“设置命令”、“测试命令”、“查询命令”和“执行命令”四种类型。

1. GSM Rec.07.07 标准 AT 命令

1) 通用命令

通用命令如表 5-2 所示。

表 5-2 通用命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
查询制造商	—	—	AT+CGMI=?	AT+CGMI
查询模块型号	—	—	AT+CGMM=?	AT+CGMM
查询模块版本	—	—	AT+CGMR=?	AT+CGMR
查询产品系列号	—	—	AT+CGSN=?	AT+CGSN
选择TE字符集	AT+CSCS=<chest>	AT+CSCS?	AT+CSCS=?	—
移动台标识请求	—	—	AT+CIMI=?	AT+CIMI
选择无线网络	AT+WS46=[<n>]	AT+WS46?	AT+WS46=?	—

2) 呼叫控制命令

呼叫控制命令如表 5-3 所示。

表 5-3 呼叫控制命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
选择地址类型	AT+CSTA=[<type>]	AT+CSTA?	AT+CSTA=?	—
移动台呼叫某一号	—	—	—	ATD[<dial_str>]
重复上次呼叫号码	—	—	—	ATDL[:]
呼叫模式	AT+CMOD=[<mode>]	AT+CMOD?	AT+CMOD=?	—
呼叫挂起	—	—	AT+CHUP=?	AT+CHUP
选择承载业务类型	AT+CBST=[<crv>]	AT+CBST?	AT+CBST=?	—
无线链路协议	AT+CRLP=[<crv>]	AT+CRLP?	AT+CRLP=?	—
业务上报控制	AT+CR=[<mode>]	AT+CR?	AT+CR=?	—
扩展错误报告	—	—	AT+CEER=?	AT+CEER
蜂窝结果码	AT+CRC=[<mode>]	AT+CRC?	AT+CRC=?	—
单一编码方案	AT+CSNS=[<mode>]	AT+CSNS?	AT+CSNS=?	—

3) 网络业务命令

网络业务命令如表 5-4 所示。

表 5-4 网络业务命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
用户号码	—	—	—	AT+CNUM
网络注册信息	AT+CREG=[<n>]	AT+CREG?	AT+CREG=?	—
选择运营商	AT+COPS=[<...>]	AT+COPS?	AT+COPS=?	—
设备锁定	AT+CLCK=[<...>, ...]	AT+CLCK?	AT+CLCK=?	—
修改密码	AT+CPWD=[<...>, ...]	AT+CPWD?	AT+CPWD=?	—
显示主叫识别	AT+CLIP=[<n>]	AT+CLIP?	AT+CLIP=?	—
显示主叫识别限制	AT+CLIR=[<n>]	AT+CLIR?	AT+CLIR=?	—
显示被叫识别	AT+COLP=[<n>]	AT+COLP?	AT+COLP=?	—
封闭用户组	AT+CCUG=[...]	AT+CCUG?	AT+CCUG=?	—
呼叫前号码与条件	AT+CCFC=...	—	AT+CCFC=?	—
呼叫等待	AT+CCWA=[...]	AT+CCWA?	AT+CCWA=?	—
呼叫保持多方通话	AT+CHLD=[<n>]	AT+CHLD?	AT+CHLD=?	—
非结构化附加业务	AT+CUSD=[...]	AT+CUSD?	AT+CUSD=?	—
计费通知	—	AT+CAOC?	AT+CAOC=?	AT+CAOC=[<mode>]
附加业务通知	AT+CSSN=[...]	AT+CSSN?	AT+CSSN=?	—
查询当前呼叫	—	—	AT+CLCC=?	AT+CLCC
优选运营商列表	AT+CPOL=[...]	AT+CPOL?	AT+CPOL=?	—
查询运营商名称	—	—	AT+COPN=?	AT+COPN

4) ME 控制和状态命令

ME 控制和状态命令见表 5-5 所示。

表 5-5 ME 控制和状态命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
手机活动状态	—	—	AT+CPAS=?	AT+CPAS
设置手机功能	AT+CFUN=[...]	AT+CFUN?	AT+CFUN=?	—
输入 PIN	AT+CPIN=[pin]>...	AT+CPIN?	AT+CPIN=?	—
电池充电	—	—	AT+CBC=?	AT+CBC
信号质量	—	—	AT+CSQ=?	AT+CSQ
选择电话簿存储	AT+CPBS=[<...>]	AT+CPBS?	AT+CPBS=?	—
查询电话簿记录	AT+CPBR=...	—	AT+CPBR=?	—
写电话簿记录	AT+CPBW=[...]	—	AT+CPBW=?	—
SIM 卡接入限制	AT+CRSM=...	—	AT+CRSM=?	—
静音控制	AT+CMUT=[<n>]	AT+CMUT?	AT+CMUT=?	—
累计呼叫计量器	AT+CACM=[...]	AT+CACM?	AT+CACM=?	—
累计呼叫最大值	AT+CAMM=[...]	AT+CAMM?	AT+CAMM=?	—
单位价格和货币表	AT+CPUC=...	AT+CPUC?	AT+CPUC=?	—
设置语音信箱号码	AT+CSVM=...	AT+CSVM?	AT+CSVM=?	—
设置事件	AT+CLAE=[<mode>]	AT+CLAE?	AT+CLAE=?	—
设置语音	AT+CLAN=[<mode>]	AT+CLAN?	AT+CLAN=?	—
查询可用 AT 命令	—	—	—	AT+CLAC
实时时钟	AT+CCLK=[<time>]	AT+CCLK?	—	—

5) ME 错误报告命令

ME 错误报告命令如表 5-6 所示。

表 5-6 ME 错误报告命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
上报移动设备错误	AT+CMEE=[...]	AT+CMEE?	AT+CMEE=?	—

2. ITU-T Rec.V25tec AT 命令

1) 通用命令

通用命令如表 5-7 所示。

表 5-7 通用命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
复位为缺省配置	—	—	—	ATZ<value>
TA的制造商信息	—	—	—	ATI
TA的制造商ID	—	—	AT+GMI=?	AT+GMI
TA模式标识	—	—	AT+GMM=?	AT+GMM
TA修改号码	—	—	AT+GMR=?	AT+GMR
请求TA序列号	—	—	AT+GSN=?	AT+GSN
TA总容量请求	—	—	AT+GCAP=?	AT+GCAP
命令行终止符	ATS3=<n>	ATS3?	ATS3=?	—
响应格式字符	ATS4=<n>	ATS4?	ATS4=?	—
编辑字符	ATS5=<n>	ATS5?	ATS5=?	—
命令回显模式	ATE<value>	—	—	—
结果码拟制	ATQ<value>	—	—	—
返回结果格式	ATV<value>	—	—	—
连接结果	ATX<value>	—	—	—
DCD使用状态	AT&C<value>	—	—	—
DTR使用状态	AT&D<value>	—	—	—
固定TE-TA数据速率	AT+IPR=<rate>	AT+IPR?	AT+IPR=?	—
TE-TA帧格式	AT+ICF=[...]	AT+ICF?	AT+ICF=?	—

2) 呼叫控制命令

呼叫控制命令如表 5-8 所示。

表 5-8 呼叫控制命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
音频拨号	—	—	—	ATT
脉冲拨号	—	—	—	ATP
呼叫应答	—	—	—	ATA
挂机控制	—	—	—	ATH
返回数据状态	—	—	—	ATO
自动应答前振铃	ATS0=<n>	ATS0?	ATS0=?	—
挂机时延	ATS10=<n>	ATS10?	ATS10=?	—

3) 数据压缩命令

数据压缩命令如表 5-9 所示。

表 5-9 数据压缩命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
V.42位数据压缩	AT+DS=[...]	AT+DS?	AT+DS=?	—
V.42位压缩报告	AT+DR=<value>	AT+DR?	AT+DR=?	—

3. 标准 GPRS AT 命令(GSM Rec.07.07)

标准 GPRS AT 命令如表 5-10 所示。

表 5-10 标准 GPRS AT 命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
定义PDP上下文	AT+CGDCONT=[...]	AT+CGDCONT?	AT+CGDCONT=?	—
请求服务质量简报	AT+CGQREQ=[...]	AT+CGQREQ?	AT+CGQREQ=?	—
可接收的服务质量	AT+CGQMIN=[...]	AT+CGQMIN?	AT+CGQMIN=?	—
GPRS附着和分离	AT+CGATT=[...]	AT+CGATT?	AT+CGATT=?	—
PDP上下文激活	AT+CGACT=...	AT+CGACT?	AT+CGACT=?	—
进入数据模式	AT+CGDATA=[...]	—	AT+CGDATA=?	—
显示PDP地址	AT+CGPADDR=[...]	—	AT+CGPADDR=?	—
应答PDP上下文请求	AT+CGAUTO=<n>	AT+CGAUTO?	AT+CGAUTO=?	—
应答PDP上下文请求	AT+CGANS=[...]	—	AT+CGANS=?	—
GPRS移动台类别	AT+CGCLASS=<class>	AT+CGCLASS?	AT+CGCLASS=?	—
GPRS事件上报	AT+CGEREP=[...]	AT+CGEREP?	AT+CGEREP=?	—
GPRS网络注册状态	AT+CGREG=<n>	AT+CGREG?	AT+CGREG=?	—
接受网络侧PDP请求	ATA	—	—	—
拒绝网络侧PDP请求	ATH	—	—	—

4. GSM Rec.07.05 命令

1) 通用配置命令

通用配置命令如表 5-11 所示。

表 5-11 通用配置命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
选择消息业务	AT+CSMS=...	AT+CSMS?	AT+CSMS=?	—
优选消息存储器	AT+CPMS=...	AT+CPMS?	AT+CPMS=?	—
SMS格式	AT+CMGF=<mode>	AT+CMGF?	AT+CMGF=?	—

2) 消息配置命令

消息配置命令如表 5-12 所示。

表 5-12 消息配置命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
服务中心地址	AT+CSCA=...	AT+CSCA?	AT+CSCA=?	—
设置文本格式参数	AT+CSMP=[...]	AT+CSMP?	AT+CSMP=?	—
显示文本格式参数	AT+CSDH=[...]	AT+CSDH?	AT+CSDH=?	—
选择小区广播消息	AT+CSCB=[...]	AT+CSCB?	AT+CSCB=?	—
保存设置	—	—	AT+CSAS=?	AT+CSAS
恢复设置	—	—	AT+CRES=?	AT+CRES[...]

3) 消息接收和读出命令

消息接收和读出命令如表 5-13 所示。

表 5-13 消息接收和读出命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
给TE指示新消息	AT+CNMI=[...]	AT+CNMI?	AT+CNMI=?	—
查询消息	AT+CMGL=[...]	—	—	—
读出消息	AT+CMGR=<index>	—	—	—
新消息确认	—	AT+CNMA?	AT+CNMA=?	AT+CNMA

4) 消息发送和写入命令

消息发送和写入命令如表 5-14 所示。

表 5-14 消息发送和写入命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
发送消息	AT+CMGS=...	—	AT+CMGS=?	—
从存储器发送消息	AT+CMSS=...	—	AT+CMSS=?	—
把消息写入存储器	AT+CMGW=...	—	AT+CMGW=?	AT+CMGW
删除消息	AT+CMGD=...	—	AT+CMGD=?	—
发送命令	AT+CMGC=...	—	AT+CMGC=?	—

5. AT 扩展命令

1) HUAWEI 命令集

HUAWEI 命令集如表 5-15 所示。

表 5-15 HUAWEI 命令集

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
网络注册业务选择	AT%NRG=[...]	AT%NRG?	AT%NRG=?	—
查询PUCT累计通话	—	—	AT%CACM=?	AT%CACM
查询PUCT当前通话	—	—	AT%CAOC=?	AT%CAOC
通话计时器值	—	—	AT%CTV=?	AT%CTV
呼叫进展信息	AT%CPI=<n>	AT%CPI?	AT%CPI=?	—
配置SIM应用工具箱	AT%SATC=...	AT%SATC?	AT%SATC=?	—
发送SAT包络命令	AT%SATE=...	AT%SATE?	AT%SATE=?	—
发送SAT命令返回值	AT%SATR=...	—	AT%SATR=?	—
终止SAT命令或通话	AT%SATT=<cs>	—	AT%SATT=?	—
GPRS字节计数器	AT%SNCNT=<rst>	AT%SNCNT?	AT%SNCNT=?	—
自动附着模式	AT%CGAATT=...	AT%CGAATT?	AT%CGAATT=?	—
加密指示	AT%CPRI=<mode>	AT%CPRI?	AT%CPRI=?	—
GPRS扩展注册状态	AT%CGREG=<mode>	AT%CGREG?	AT%CGREG=?	—
测试SIM卡是否存在	—	—	—	AT%TSIM
上次通话时长	—	—	—	AT%LCD
总通话时长	—	—	—	AT%TCD
模块关机	—	—	—	AT%MSO
查询消息	—	—	—	AT%MGL
读出消息	AT%MGR=<index>	—	AT%MGR=?	—
休眠控制	AT%SLEEP=<mode>	AT%SLEEP?	—	—

2) 音频设置相关命令集

音频设置相关命令集如表 5-16 所示。

表 5-16 音频设置相关命令集

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
设置音频输入参数	AT%NFI=...	AT%NFI?	AT%NFI=?	—
设置音频输出参数	AT%NFO=...	AT%NFO?	AT%NFO=?	—
设置扬声器音量	AT%NFV=<vol>	AT%NFV?	AT%NFV=?	—
保存音频配置参数	AT%NFW=<mode>	AT%NFW?	AT%NFW=?	—
选择音频配置参数	AT%NFS=<mode>	AT%NFS?	AT%NFS=?	—
设置测音音量	AT%STN=<vol>	AT%STN?	AT%STN=?	—
回声抑制	AT%VLB=<enable>	AT%VLB?	AT%VLB=?	—
音频通道选择	AT%SNFS=<path>	—	AT%SNFS=?	—

6. TCP/IP AT 命令

1) 初始化命令

初始化命令如表 5-17 所示。

表 5-17 初始化命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
初始化命令	AT%ETCPIP=[...]	AT%ETCPIP?	AT%ETCPIP=?	—

2) 打开链接命令

打开链接命令如表 5-18 所示。

表 5-18 打开链接命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
打开一条TCP或UDP	AT%IPOPEN=...	AT%IPOPEN?	AT%IPOPEN=?	—
打开多条TCP或UDP	AT%IPOPENX=...	AT%IPOPENX?	AT%IPOPENX=?	—

3) 设置数据模式命令

设置数据模式命令如表 5-19 所示。

表 5-19 设置数据模式命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
设置数据模式命令	AT%IOMODE=[...]	AT%IOMODE?	AT%IOMODE=?	—

4) 数据发送命令

数据发送命令如表 5-20 所示。

表 5-20 数据发送命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
单链接下发送数据	AT%IPSEND=<data>	AT%IPSEND?	—	—
多链接下发送数据	AT%IPSENDX=<data>	—	—	—

5) 关闭链接命令

关闭链接命令如表 5-21 所示。

表 5-21 关闭链接命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
关闭链接命令	AT%IPCLOSE=[...]	AT%IPCLOSE?	AT%IPCLOSE=?	—

6) TCP 的 ACK 查询或清除命令

TCP 的 ACK 查询或清除命令如表 5-22 所示。

表 5-22 TCP 的 ACK 查询或清除命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
ACK 查询或清除	AT%TXSTATE=[<...>]	AT%TXSTATE?	AT%TXSTATE=?	—

7) 数据操作命令

数据操作命令如表 5-23 所示。

表 5-23 数据操作命令

名 称	设置命令形式	查询命令形式	测试命令形式	执行命令形式
数据查询命令	—	—	—	AT%IPDQ
数据读取命令	AT%IPDR=[<index>]	—	—	—
数据删除命令	AT%IPDD[...]	—	—	—
数据删除模式命令	AT%IPDDMODE=<mode>	AT%IPDDMODE?	AT%IPDDMODE=?	—

5.1.4 应用电路与编程

目前,在车载导航、GPS 定位和远程数据传输等应用中,往往通过 GTM900 无线模块(或 TC35i)实现“数据”的远程通信。图 5-7 所示为将现场“温度信息”和其他“物理信息”由 C8051 单片机实时处理,通过“GTM900 无线模块”实现远程测量和传送。

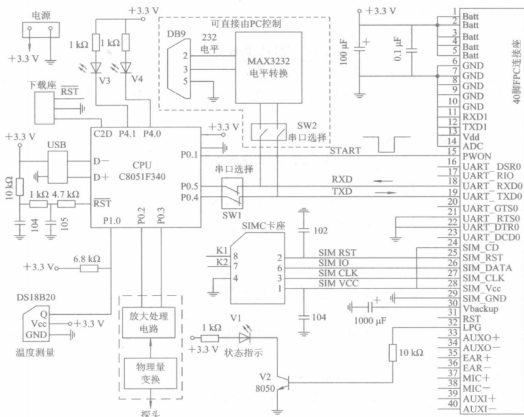


图 5-7 GTM900 的应用电路

在这个应用中,还可通过 SW1 与 SW2 的开关位设置成直接由 PC 机控制的“无线”传输设备。

只要“手机”能够通信的地方,这种测量传输就完全能够使用。

下面通过“短信方式”和“GPRS 上网方式”,使用 AT 命令完成“点对点”的通信。

```
#include "C8051F340.h"          /*C8051F340 的头文件*/
#include <string.h>              /*使用了里面的 strlen()函数*/
sbit PWON = P0^1;               /* GTM900 复位管脚*/

unsigned char code Com mAnd_CMGF[]={ "AT+CMGF=0\r"; /*设置 PDU 模式,发送短信用*/
unsigned char code Com mAnd_CNMI[]={ "AT+CNMI=2,2\r"; /*设置短信接收模式*/
/*直接显示短消息内容*/

unsigned char code Com mAnd_ATE0[]={ "ATE0\r"; /*关闭回显*/
unsigned char code Com mAnd_CMGS[]={ "AT+CMGS="; /*发送长度*/
unsigned char code Com mAnd_CMGR[]={ "AT+CMGR="; /*读第几条短信*/
unsigned char code Com mAnd_CMGD[]={ "AT+CMGD="; /*删除第几条短信*/
unsigned char code Com mAnd_CGDCONT[]={ "AT+CGDCONT=1,\"IP\",\"CMWAP\r";
/*连接到 WAP 网上,要连接到互联网上用"CMNET"*/

unsigned char code Com mAnd_ETCPIP[]={ "AT+ETCPIP\r"; /*进入 TCP/IP 功能*/
unsigned char code Com mAnd_IPOPEN[]={ "AT+IOPEN=\\"TCP\\","\010.000.000.172
\","23\r"; /*以 TCP 方式连接到 10.0.0.172:23 上面*/

unsigned char code Send_SMS_Number[]={ "13500000000"; /*目的手机号码*/
unsigned char SMS_Change_Number[13]; /*转换格式后保存的手机号码*/
/*关闭看门狗和使用外部晶振*/
void System_Init(void)
{
    PCA0MD    &= ~0x40; /*关闭看门狗*/
    OSCICN    |= 0x03; /*使用内部晶振,不分频*/
}

/*****端口配置*****/
void Port_Init(void)
{
    P0MDIN = 0x3f; /*晶振端口为模拟量*/
    P0SKIP = 0xCf; /*使 P0.4、P0.5 作为串口 0 的管脚*/
    P0MDOUT = 0x00;
    XBR0    = 0x01;
    XBR1    = 0x40; /*串口 0 允许*/
}

/**系统时钟选择**/
void OSCILLATOR_Init(void)
{
    unsigned int i;
    OSCXCN = 0x67; /*晶体振荡方式, 12 MHz*/
```

```

    for(i = 0; i < 1000; i++); /*延时 1 ms*/
    while (!(OSCXCN & 0x80)); /*等待晶体振荡稳定*/
    CLKSEL = 0x01; /*系统时钟切换至外部振荡器*/
    OSCICN = 0x00; /*关闭内部振荡器*/
}
/*****UART0 初始化函数*****/
void UART0_Init(void) /*使用外部晶振 22.1184 MHz 产生 9600 b/s 的波特率*/
{
    TMOD &= 0x0f; /*选择 T1 工作模式*/
    TMOD |= 0x20;
    SCON0 = 0x10;
    CKCON = 0x00; /*sysclk/12*/
    TH1 = 160; /*sysclk/(256-th1)*0.5=baud..160:9600 b/s 64:4800 b/s*/
    TR1 = 1;
}
/****短延时函数****/
void DelayUs(unsigned int u)
{
    unsigned int i;
    while(u--)
        for(i=0; i<1000; i++);
}
/****长延时函数****/
void DelayMs(unsigned int m)
{
    while(m--)
        DelayUs(1000);
}
/****发送一个字节****/
void SMSsendbyte(unsigned char p)
{
    SBUF0=p;
    while(TI0==0);
    TI0=0;
    DelayUs(1);
}
/*****
功能：把 00H~0FH 的十六进制数转换为 ASCII 码
入口参数：十六进制数据
返回参数：转换后的 ASCII 码(0AH~0FH 转换后得到大写字母)
*****/
unsigned char toASCII(unsigned char ch)

```

```

    {
        if( (ch>=0x00)&&(ch<=0x09) )
            ch+=0x30;
        else if( (ch>=0x0A)&&(ch<=0x0F) )
            ch+=0x37;    /*按大写字母发送*/
        return(ch);
    }

/*-----
功能：把一个字节的数以 PDU 格式发送
入口参数：十六进制数据
返回参数：无
-----*/
void toPDU_Send(unsigned char ch)
{
    unsigned char a,b;
    a=ch/16;
    b=ch%16;
    SMSsendbyte(toASCII(a));
    SMSsendbyte(toASCII(b));
}

/***发送字符串***/
void SMSsendstring(unsigned char *p)
{
    unsigned int len,i;
    len=strlen(p);
    for(i=0; i<len; i++)
        SMSsendbyte(p[i]);
}

/***等待 GTM900 返回回答命令，ch 为一个字节的命令***/
void WaitGTM900(unsigned char ch)
{
    {
        do{
            while(RIO==0);
            RIO=0;
        }
        while(SBUF0!=ch);
    }
}

/***GPRS 连接方式***/
void GPRS_connect(void)
{
    SMSsendstring(Com mAnd_IOPEN); /*这里采用 TCP 方式，模块还支持 UDP 方式连接*/
}

```



```

WaitGTM900(T); /*等待模块返回"CONNECT"回答命令*/
}
/*-----
功能: GTM900 初始化, PDU 模式、短信接收模式通过直接写串口来配置网络
指令: ATE0 回车
      AT+CMGF=0 回车
      AT+CNMI=2,2 回车
      AT+CGDCONT=1,"IP","CMWAP"回车
      AT%ETCPIP 回车
-----*/

void SMSInit(void)
{
    PWON = 0;
    DelayMs(10);
    PWON = 1; /*复位脉冲, 低电平延时 10 ms 左右*/
    WaitGTM900("T"); /*等待 GTM900 返回命令*/
    DelayMs(20); /*模块进入正常工作前需延时*/
    /*发送初始化指令*/
    /*关闭回显*/
    SMSsendstring(Com mAnd_ATE0);
    WaitGTM900("K");
    DelayUs(100);
    /*设置 PDU 模式*/
    SMSsendstring(Com mAnd_CMGF);
    WaitGTM900("K");
    DelayMs(6);
    /*设置短信接收模式*/
    SMSsendstring(Com mAnd_CNMI);
    WaitGTM900("K");
    DelayMs(6);
    /*上网方式*/
    SMSsendstring(Com mAnd_CGDCONT);
    WaitGTM900("K");
    DelayUs(100);
    SMSsendstring(Com mAnd_ETCPIP);
    WaitGTM900("K");
    DelayUs(100);
}
/*-----
功能: 把 PDU 格式还原为 HEX 码
入口参数: pdu_H、pdu_L 分别为 PDU 格式的高位和低位

```

返回参数: 转换后的 HEX 格式的数

```

-----*/
unsigned char PDU_to_HEX(unsigned char pdu_H,unsigned char pdu_L)
{
    unsigned char a,b;
    if(pdu_H>='A')
        a=(pdu_H-'A'+10)*16;
    else a=(pdu_H-'0')*16;
    if(pdu_L>='A')
        b=(pdu_L-'A'+10);
    else b=(pdu_L-'0');
    return(a+b);
}
/*-----

```

功能: 正常顺序的字符串转换为两两颠倒的字符串, 若长度为奇数, 则补“F”凑成偶数, 如
“13512345678” -> “3115325476F8”

入口参数: Snum 为源字符串指针, Dnum 为目标字符串指针

SnumLength 为源字符串长度

返回参数: 目标字符串长度

```

-----*/
int GsmInvertNumbers(* Snum, * Dnum, SnumLength)
unsigned char* Snum, unsigned char* Dnum, unsigned char SnumLength;
{
    unsigned char i;
    unsigned char DnumLength;      /*目标字符串长度*/
    unsigned char ch;              /*用于保存一个字符*/
    DnumLength = SnumLength;       /*复制串长度*/
    for(i=0; i<SnumLength; i+=2)  /*两两颠倒*/
    {
        ch = *Snum++;              /*保存先出现的字符*/
        *Dnum++ = *Snum++;         /*复制后出现的字符*/
        *Dnum++ = ch;              /*复制先出现的字符*/
    }
    if(SnumLength & 1)             /*源串长度是否为奇数? */
    {
        *(Dnum-2) = 'F';          /*补"F" */
        DnumLength++;             /*目标串长度加 1*/
    }
    *Dnum = '\0';                  /*输出字符串加个结束符*/
    return DnumLength;             /*返回目标字符串长度*/
}

```

```

/**短信形式发送“测试”两字到目的手机号码上**/
void Send_SMS(void)
{
    SMSsendstring(Com mAnd_CMGS); /*发送短信的命令头*/
    SMSsendstring("19\r");
    /*发送“测试”两字，占4个字节，加上系统的15个字节*/
    WaitGTM900('>'); /*等待模块返回 >*/
    DelayUs(20);
    SMSsendstring("0011000D9168");
    GsmInvertNumbers(Send_SMS_Number,SMS_Change_Number,11); /*号码格式转换*/
    SMSsendstring(SMS_Change_Number); /*要发送的手机号码码*/
    SMSsendstring("0008FF04"); /*最后的"04"表示要发送四个字节*/
    SMSsendstring("6D4B8BD5"); /*PDU 格式，内容为“测试”*/
    DelayUs(100);
    SMSsendbyte(0x1A); /*<ctrl-z>符号*/
    WaitGTM900('K'); /*等待发送成功*/
}

void main(void) /*测试函数*/
{
    System_Init();
    Port_Init(); /*端口初始化*/
    OSCILLATOR_Init(); /*晶振初始化*/
    UART0_Init(); /*串口0初始化*/
    SMSInit(); /*短信初始化*/
    Send_SMS(); /*短信发送函数。发送“测试”两字到指定号码*/
    GPRS_connect(); /*GPRS 连接，以 TCP 方式连接到 10.0.0.172:23 上*/
    /*连接成功后，就可以用 AT%IPSEND 与对方通信了*/
    /*格式为 AT%IPSEND=<data>*/
    /*通信完后可用 AT%IPCLOSE=1 命令关闭此连接*/
    //如果读者想接收短信或者接收 GPRS 数据，则建议在串口 0 中断中去处理，这里不加以介绍
    while(1); /*等待*/
}

```

5.2 JZ87x 系列无线数传通信模块

5.2.1 硬件与功能描述

JZ87x 系列数传模块基于 GFSK 的调制方式，传输(发射)功率为 10 mW~10 W，传输空旷距离为 300 m~10 km。JZ878 实际使用中至少可以穿透多座 12 层钢筋混凝土结构的楼房，可以很好地满足普通的使用要求。

JZ87x 系列采用透明的数据传输，所发即所收。在使用过程中用户加入自己的通信协议，

无线模块则不对传递的数据做任何修改和解析。在正常通信的情况下,良好的协议与硬件的畅通就可保证数据的连续性和正确性。

JZ87x 系列提供多个信道,可满足多种通信组合方式的需求。每组使用不同的信道进行通信,以避免数据包的冲突和保证通信的及时性。无线模块可在 1200~38 400 b/s 等多种波特率下工作,且无线传输速率与接口波特率成正比,以满足终端设备对多种波特率的需要。

1. 主要性能特点

- (1) 系列模块视距可靠传输距离为 300 m~10 km。
- (2) 载频 433 MHz,也可订制 315/868/915 MHz 等其他载频。
- (3) 提供 16 个信道,如果用户需要,则可扩展到 32 信道。
- (4) 接口速率: 1200/2400/4800/9600/19 200/38 400 b/s。
- (5) 信道速率: 1200/2400/4800/9600/19 200/38 400 b/s。
- (6) 接口方式: TTL/RS232/485 由用户选择。接口格式为 8N1/8E1/8o1。
- (7) 数据收发转换自动完成,只要向接口收/发数据即可,转换时间短。
- (8) 可用于点对点、点对多点、多点点对等多种通信组合方式。
- (9) 数据透明传输,可传输较长的数据帧。
- (10) 自动过滤掉空中产生的假数据,长期使用可靠性好,故障率极低。
- (11) 功耗:接收电流<45 mA,发射电流<1.5 A(JZ878)。
- (12) 三种省电模式:硬件唤醒、串口唤醒、空中唤醒。
- (13) 供电:DC +4.5~+5.5 V。对于 JZ878 来说,为+12 V。
- (14) 温度: -20~+65℃。
- (15) 互通型号: JZ871、JZ872、JZ873、JZ875、JZ878。

2. 组成框图与接口

JZ87x 系列数传模块的组成框图如图 5-8 所示。



图 5-8 JZ87x 系列数传模块的组成框图

3. 通信接口

JZ87x 系列模块提供了 TTL/RS232/485 三种接口。用户只需插拔短路器再上电即可改变接口类型。对于在同一电路板上的通信,通常选 TTL 电平的串口;对于通信距离在 100 m 以内的通信,选择 RS232 电平较合适;对于大于 100 m 小于 1000 m 的通信最好采用 485 接口。

通信距离超过 100 m 就达到了 RS232 的传输极限, 超过 1200 m 就达到了 RS485 的传输极限。当超过传输极限时, 一般采用中继器或者转换头的处理方式, 费用会有一定的增加, 布线也比较复杂, 需要专业人士合理布线才能很好地工作, 而且随着距离的增加, 传输延迟也会相应增加。如果距离比较远, 则最好选用通信距离远一点的模块, 如 JZ878 就可传输 5 km 以上。

在选择波特率时, 也要根据当时的使用条件选择, 一般通道(空中)速率比接口速率大一点较好, 波特率越低, 传输距离越远。

5.2.2 通信协议的构建

通过 JZ87x 系列模块进行数据无线传输时, 通常要有传输协议。传输协议的好坏会影响传输结果。介于无线模块的传输特性和易受外界干扰等特点, 在数据较大(一般大于 256 字节)时, 最好将大数据包分成若干个小包传输。每个数据包均加“校验字节”, 如果有错, 则重收/发有问题的“数据包”就可以很好地解决传输问题。

表 5-24 是常用的通信协议格式。主发(甲)方和从收(乙)方的通信格式是:

帧头 + 命令 + 设备号 + 数据长度 + 包数 + 数据块 + 校验字节

表 5-24 常用的通信协议格式

甲 方(主)		乙 方(从)	
帧头(0xAA)	一个字节	帧头(0xBB)	一个字节
帧头(0x75)	一个字节	帧头(0x7A)	一个字节
命令	一个字节	命令	一个字节
设备号	一个字节	设备号	一个字节
数据长度	一个字节	数据长度	一个字节
包数	一个字节	包数	一个字节
备份	一个字节	备份	一个字节
数据块……	N个字节	数据块……	N个字节
校验字节	一个字节	校验字节	一个字节

其中, 帧头占两个字节, 命令占一个字节, 设备(通道)号占一个字节, 数据长度占一个字节, 包数占一个字节, 备份占一个字节, 数据块占若干字节(最多 256 字节), 校验占一个字节。

校验字节可采用从帧头开始到校验字节前的数据“累加和”(舍去高字节, 只用最低字节), 也可用从帧头开始到校验字节前的数据“异或值”。

5.2.3 应用电路与编程

图 5-9 (a)是由 STC12C4052AD 单片机与 JZ873 构成的远程温度/湿度采集系统。图 5-9 (b)接收图 5-9(a)发出的数据, 其接收模块也用 JZ873。

作为一个实例, 用表 5-24 所示的数据传输格式规定图 5-9 (b)是甲方, 图 5-9 (a)是乙方。用 SHT10 传感器测量温湿度可用 6 个字节(用 BCD)表示, 假设数据为 23.45℃, 75.31%,

即 002345, 007531。表 5-25 是传输的数据格式。

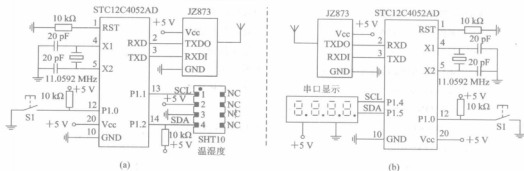


图 5-9 由 JZ873 实现的异地通信

(a) 测量发送电路; (b) 远程接收电路

表 5-25 传输的数据格式

甲方要乙方数据的“数据帧”		乙方回送给甲方的“数据帧”	
标准格式	实际数据	标准格式	实际数据
帧头(0xAA)	0xAA	帧头(0xBB)	0xBB
帧头(0x75)	0x75	帧头(0x7A)	0x7A
命令	0x10	命令	0x11
设备号	1	设备号	1
数据长度	0	数据长度	6
包数	1	包数	1
备份	0	备份	0
数据块……	0	数据块……	0x002345007531
校验字节	0x31(累加和)	校验字节	0x5C(累加和)

设波特率为 4.8 kb/s(接口速率), 8 位数据无奇偶校验(共 10 位)。

甲方在发出“0xAA、0x75、0x10、0x01、0、0、0、0、0x31”时, 乙方在收到该“数据帧”后, 马上回送“0xBB、0x7A、0x11、0x01、0x06、0、0、0x002345007531、0x5C”, 甲方就能收到该数据包了。

用 C51 所实现的源程序有两部分, 即甲方程序部分和乙方程序部分。

```

/*****甲方程序部分*****/
/*甲方请求向乙方数据*/

#include <reg52.h>
#define JS MAX 0x450000 /*超时设置*/
#define S_AA 0xAA /*发送帧头 1*/

```

```

#define S_75    0x75                /*发送帧头 2*/
#define R_BB    0xBB                /*接收帧头 1*/
#define R_7A    0x7A                /*接收帧头 2*/
sbit key=P1^0;                       /*定义按键*/
unsigned char idata x_data[0x15];    /*数据区*/
void out_sbuf(void)                  /*串口初始化*/
{
    SCON=0x50;
    TMOD |=0x21;                    /*4.8 kb/s 波特率*/
    PCON=0x00;
    TL1=0xfa;
    TH1=0xfa;
    TR1=1;
}
void delay_s(unsigned int k)          /*短延时函数, k 是时常数*/
{
    unsigned int i;
    for (i=0; i<k; i++);
}
/*-----由串口发单个字符-----*/
void sendchar(unsigned char ch)
{
    SBUF=ch;
    while(TI==0);
    TI = 0;
    delay_s(10);                    /*延时*/
}
/*有条件地读一个字符*/
unsigned char gethex_H(void)           /*超时退出*/
{
    unsigned long i=0;
    unsigned char c=0;
    while((RI==0)&&(i<JS MAX)) i++; /*超时增 1*/
    c = SBUF;
    RI = 0;
    return(c);
}
/*cd 是命令, type 是设备号, leth 是长度, numb 是包数*/
void send_wdnumb(cd,type,leth,numb) /*发送数据函数*/
    unsigned char cd,type,leth,numb;
{

```

```

unsigned char i,sum=0;
sendchar(S_AA);          /*发送帧头 AA*/
sum= sum+S_AA;
sendchar(S_75);          /*发送帧头 75*/
sum= sum+S_75;
sendchar(cd);            /*发送命令*/
sum= sum+cd;
sendchar(type);          /*发送设备号*/
sum= sum+type;
sendchar(leth);          /*发送长度*/
sum= sum+leth;
sendchar(numb);          /*发送包数*/
sum= sum+numb;
sendchar(0);             /*发送备份 0*/
sum= sum+0;
if(leth==0)
{
    sendchar(0);          /*发送数据 0*/
    sum= sum+0;
}
else
{
    for(i=0; i<leth; i++)
    {
        sendchar(x_data[i]); /*发送数据*/
        sum= sum+x_data[i];
    }
}
sendchar(sum);           /*发送校验和*/
}

/*-----接收数据---返回 0 正确, 返回 1 出错-----*/
unsigned char get_data(void) /*接收的数据在 x_data[]中*/
{
    unsigned char i,c_data,leth,sum=0;
    c_data=gethex_H();      /*接收帧头 0xBB*/
    sum = sum+c_data;        /*计算校验数据*/
    if (c_data !=R_BB) goto end_getdata;
    c_data=gethex_H();      /*接收帧头 0x7A*/
    sum = sum+c_data;        /*计算校验数据*/
    if (c_data !=R_7A) goto end_getdata;

```



```

c_data=gethex_H();          /*接收命令*/
sum = sum +c_data;          /*计算校验数据*/
x_data[0]=c_data;          /*存储命令*/
c_data=gethex_H();          /*接收设备号*/
x_data[1]=c_data;          /*存储设备号*/
sum = sum +c_data;          /*计算校验数据*/
c_data=gethex_H();          /*接收长度*/
sum = sum +c_data;          /*计算校验数据*/
x_data[2]=c_data;          /*存储长度*/
lenth=c_data+2;            /*长度+2*/
for(i=0; i<lenth; i++)
{
    x_data[i+3]=gethex_H();  /*接收包数、备份、数据、校验和*/
    sum = sum +c_data;      /*计算校验数据*/
}
c_data=gethex_H();          /*校验和数据(最后一个数据)*/
if(sum ==c_data) return(0) ; /*返回数据 0 表示数据正确*/
end_getdata:
return(1) ;                 /*返回数据 1 表示数据不正确*/
}

void main(void)              /*甲方主函数部分*/
{ unsigned char k,i,result[0x15];
  out_sbuf();                /*串口初始化*/
  while (1)
  {
    if(key==0)                /*有按键请求数据*/
    {
      /*甲方请求乙方温度/湿度数据*/
      send_wdnumb(0x10,0x01,0,1); /*AA7510010001000031H*/
      k=get_data();            /*接收的数据在 x_data[]中*/
      if(k==0)                 /*接收成功*/
      {
        if(x_data[0]==0x11)    /*0x11 是返回的温度/湿度值*/
        {
          for(i=0; i<x_data[2]; i++)
          {
            result[i]=x_data[i+3];
            /*display(); */ /*显示温度湿度值*/
          }
        }
      }
    }
  }
}

```

```

    }

    else
    {
        /*其他命令的处理*/
    }

    /*key=0 end*/
}/*key=0 end*/
}/*while_end*/
}

/*****乙方程序部分*****/
/*乙方接收甲方命令，并向甲方回送数据*/

#include <reg51.h>

#define JS_MAX 0x45000 /*超时设置*/
#define S_BB 0xBB /*发送帧头 1*/
#define S_7A 0x7A /*发送帧头 2*/
#define R_AA 0xAA /*接收帧头 1*/
#define R_75 0x75 /*接收帧头 2*/

sbit key=P1^0; /*定义按键*/
sbit scl=P1^1; /*I2C 时钟*/
sbit sda=P1^2; /*I2C 数据*/

unsigned char idata x_data[0x15],result[0x15]; /*数据区*/
void out_sbuf(void) /*串口和中断初始化*/
{
    SCON=0x50;
    TMOD|=0x21; /*4.8 kb/s 波特率*/
    PCON=0x00;
    TL1=0xfa;
    TH1=0xfa;
    TR1=1;
    ES=1; /*串口允许中断*/
    EA=1;
}

void delay_s(unsigned char k) /*短延时函数 k 是时常数*/
{
    unsigned char i;
    for (i=0; i<=k; ++i);
}

/*-----由串口发单个字符-----*/
void sendchar(unsigned char ch)
{
    SBUF=ch;

```

```

while(TI==0);
TI = 0;
delay_s(10);          /*延时*/
}
/*有条件地读一个字符*/
unsigned char gethex_H(void)          /*超时退出*/
{
    unsigned int i=0;
    unsigned char c=0;
    while ((RI==0)|(i<JS_MAX))    i++;    /*超时增1*/
    c = SBUF;
    RI = 0;
    return(c);
}

/*cd 是命令, type 是设备号, leth 是长度, numb 是包数*/
void send_wdnumb(cd,type,leth,numb)    /*发送数据函数*/
{
    unsigned char i,sum=0;
    sendchar(S_BB);          /*发送帧头 BB*/
    sum=sum+S_BB;
    sendchar(S_7A);          /*发送帧头 7A*/
    sum=sum+S_7A;
    sendchar(cd);          /*发送命令*/
    sum=sum+cd;
    sendchar(type);          /*发送设备号*/
    sum=sum+type;
    sendchar(leth);          /*发送长度*/
    sum=sum+leth;
    sendchar(numb);          /*发送包数*/
    sum=sum+numb;
    sendchar(0);          /*发送备份 0*/
    sum=sum+0;
    if(leth==0)
    {
        sendchar(0);          /*发送数据 0*/
        sum=sum+0;
    }
    else {
        for(i=0; i<leth; i++)
        {

```

```

        sendchar(result[i]);          /*发送数据*/
        sum=sum+result[i];
    }
}

sendchar(sum);                      /*发送校验和*/
}

/*-----接收数据---返回 0 正确, 返回 1 出错-----*/
unsigned char get_data(void)          /*接收的数据在 x_data[]中*/
{
    unsigned char i,c_data,leth,sum=0;

    c_data=gethex_H();               /*接收帧头 0xAA*/
    sum=sum+c_data;                   /*计算校验数据*/
    if (c_data !=R_AA) goto end_getdata;

    c_data=gethex_H();               /*接收帧头 0x75*/
    sum=sum+c_data;                   /*计算校验数据*/
    if (c_data !=R_75) goto end_getdata;

    c_data=gethex_H();               /*接收命令*/
    sum=sum+c_data;                   /*计算校验数据*/
    x_data[0]=c_data;                 /*存储命令*/
    c_data=gethex_H();               /*接收设备号*/
    x_data[1]=c_data;                 /*存储设备号*/
    sum=sum+c_data;                   /*计算校验数据*/
    c_data=gethex_H();               /*接收长度*/
    sum=sum+c_data;                   /*计算校验数据*/
    x_data[2]=c_data;                 /*存储长度*/
    leth=c_data+2;                   /*长度+2*/
    for(i=0; i<leth; i++)
    {
        x_data[i+3]=gethex_H();      /*接收包数、备份、数据、校验和*/
        sum=sum+c_data;              /*计算校验数据*/
    }
    c_data=gethex_H();               /*校验和数据(最后一个数据)*/
    if(sum==c_data) return(0);        /*返回数据 0 表示数据正确*/
    end_getdata:
    return(1);                       /*返回数据 1 表示数据不正确*/
}

void ready_data(void)                /*准备好的数据*/
{
    result[0]=0x00; result[1]=0x23; result[2]=0x45;
    result[3]=0x00; result[4]=0x75; result[5]=0x31;
}

```

```

    }
    void sbuf_data(void) interrupt 4      /*串口中断函数*/
    { unsigned char c;
      ES=0;                             /*关中断*/
      if(TI==1) goto quit_sbuf;
      c=get_data();                     /*得到对方的命令帧*/
      if(c!=0) goto quit_sbuf;
      if(x_data[0]==0x10)
        send_wdnumb(0x11,0x01,0x06,0x01); /*(cd,type,leth,numb)*/
      quit_sbuf;
      ES=1;                             /*开中断*/
    }
    void main(void)                     /*乙方主函数部分接收命令用中断方式*/
    { out_sbuf();                       /*串口初始化*/
      while (1)
      {
        /*cj_wdspd(); */              /*采集温度、湿度*/
        ready_data();                 /*要发送的数据*/
        if(key==0)                    /*有按键立即发送数据*/
        {
          /*乙方主动向甲方发数据*/
          ES=0;                       /*关串口中断*/
          send_wdnumb(0x11,0x01,6,1); /*发送 BB7A110106010023450075315CH*/
          ES=1;                       /*开串口中断*/
        }
        /*key=0 end*/
      }
    }
  }
  /*while_end*/
}

```

5.3 nRF24L01 单片 2.4G 超低功耗数传器件

5.3.1 硬件与功能描述

nRF24L01 是一款工作在 2.4~2.5 GHz 世界通用 ISM 频段的单片无线收发器芯片。无线收发器包括：频率发生器、增强型 ShockBurst™ 模式控制器、功率放大器、晶体振荡器、调制器、解调器。输出功率、频道选择和协议可以通过 SPI 接口进行设置。

nRF24L01 的电流消耗极低。当工作在发射模式时，发射功率为 -6 dBm，电流消耗为 9.0 mA；当工作在接收模式时，电流消耗为 12.3 mA。掉电模式和待机模式下电流消耗更低 (900 nA)。

在增强型 ShockBurst™ 模式下，当 nRF24L01 工作在应答模式时，快速的空中传输及启动时间极大地降低了电流消耗。低成本 nRF24L01 集成了所有高速链路层操作，比如重发丢

失数据包和产生应答信号。SPI 接口可以利用单片机通用 I/O 口进行模拟时序。由于空中传输时间很短,极大地降低了无线传输中的碰撞现象,加上链路层完全集成在芯片上,因此非常便于软/硬件的开发。

nRF24L01 在接收模式下可以接收 6 路不同通道的数据,每一个数据通道使用不同的地址,但是共用相同的频道。也就是说,6 个不同的 nRF24L01 设置为发送模式后可以与同一个设置为接收模式的 nRF24L01 进行通信,而设置为接收模式的 nRF24L01 可以对这 6 个发射端进行识别。数据通道 0 是唯一一个可以配置为 40 位自身地址的数据通道。1~5 数据通道都为 8 位自身地址和 32 位公用地址。所有的数据通道都可以设置为增强型 ShockBurst™ 模式。

nRF24L01 在确认收到数据后记录地址,并以此地址为目标地址发送应答信号。在发送端,数据通道 0 用来接收应答信号,因此,数据通道 0 的接收地址要与发送端地址相等以确保接收到正确的应答信号。

nRF24L01 配置为增强型 ShockBurst™ 发送模式时,只要 MCU 有数据要发送,nRF24L01 就会启动 ShockBurst™ 模式来发送数据。在发送完数据后,nRF24L01 转到接收模式并等待终端的应答信号。如果没有收到应答信号,则 nRF24L01 将重发相同的数据包,直到收到应答信号或重发次数超过 SETUP_RETR_ARC 寄存器中设置的值为止,如果重发次数超过了设定值,则产生 MAX_RT 中断。

只要收到确认信号,nRF24L01 就认为最后一包数据已经发送成功(接收方已经收到数据),把 TX FIFO 中的数据清除掉并产生 TX_DS 中断(IRQ 引脚置高)。

该器件可广泛应用于无线鼠标、无线键盘、游戏机操纵杆、无线门禁、无线数据通信、安防系统、遥控装置、遥感勘测、智能运动设备、工业传感器和玩具等产品。

1. 主要性能特点

- (1) 真正的 GFSK 单收发芯片,内置链路层;
- (2) 增强型 ShockBurst™,自动应答及自动重发功能;
- (3) 地址及 CRC 检验功能;
- (4) 数据传输率为 1 或 2 Mb/s, SPI 接口数据速率为 0~8 Mb/s;
- (5) 125 个可选工作频道,很短的频道切换时间,可用于跳频;
- (6) 与 nRF24xx 系列器件完全兼容;
- (7) 可接收 5 V 电平的输入;
- (8) 极低的晶振要求,即 $\pm 60 \times 10^{-6}$;
- (9) 低成本电感和双面 PCB 板;
- (10) 20 脚 QFN 4 mm × 4 mm 封装;
- (11) 工作电压为 1.9~3.6 V;
- (12) 温度范围为 -40~+85℃。

2. 内部结构与引脚说明

nRF24L01 的内部结构与引脚排列如图 5-10 所示。

该器件采用 20 脚 QFN 封装,见图 5-10(b)。其引脚功能如表 5-26 所示。

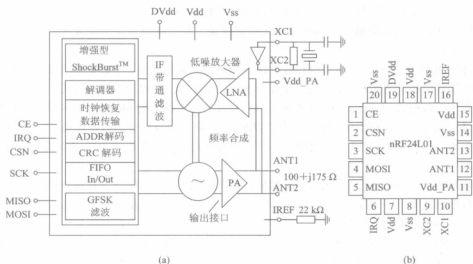


图 5-10 nRF24L01 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

表 5-26 nRF24L01 的引脚功能

引 脚	名 称	引脚功能	功 能 描 述
1	CE	数字输入	RX或TX模式选择端
2	CSN	数字输入	SPI片选信号, 低电平有效
3	SCK	数字输入	SPI时钟输入脚
4	MOSI	数字输入	SPI串行数据输入脚
5	MISO	数字输出	SPI串行数据三态输出脚
6	IRQ	数字输出	可屏蔽中断脚, 低电平有效
7,15,18	Vdd	电源	电源(+3 V)
8,14,17,20	Vss	电源	接地端(0 V)
9	XC2	模拟输出	接晶体振荡器
10	XC1	模拟输入	接晶体振荡器/外部时钟输入脚
11	Vdd_PA	电源输出	给RF的功率放大器提供的+1.8 V电源端
12	ANT1	天线	天线接口1
13	ANT2	天线	天线接口2
16	IREF	模拟输入	参考电流
19	DVdd	电源输出	去耦电路电源正极端

3. 工作过程与功能描述

1) 工作方式

nRF24L01 可以设置为表 5-27 所示的几种工作模式。

表 5-27 工作模式

模式	PWR_UP	PRIM_RX	CE	FIFO寄存器状态
接收模式	1	1	1	—
发送模式	1	0	1	数据在TX FIFO寄存器中
发送模式	1	0	1→0(“1”>10 μs)	停留在发送模式直至数据发送完
待机模式 II	1	0	0	TX FIFO为空
待机模式 I	1	—	0	无数据传输
掉电模式	0	—	—	—

(1) 待机模式。待机模式 I 是在保证快速启动的同时,减少系统平均消耗电流。在待机模式 I 下,晶振正常工作。在待机模式 II 下,部分时钟、缓冲器处在工作模式。当发送端 TX FIFO 寄存器为空并且 CE 为高电平时,进入待机模式 II。在待机模式期间,寄存器配置字的内容保持不变。

(2) 掉电模式。在掉电模式下,nRF24L01 的各项功能关闭,保持电流消耗最小。进入掉电模式后,nRF24L01 停止工作,但寄存器内容保持不变,掉电模式由寄存器中 PWR_UP 位来控制。

(3) 数据包处理方式。nRF24L01 有(ShockBurstTM 和增强型 ShockBurstTM)两种数据包处理方式。

① ShockBurstTM 模式。ShockBurstTM 模式(与 nRF2401、nRF24E1、nRF2402 和 nRF24E2 数据传输率为 1 Mb/s 时相同)下,nRF24L01 可以与成本较低的低速 MCU 相连。高速信号处理是由芯片内部的射频协议处理的,nRF24L01 提供 SPI 接口,数据率取决于单片机本身的接口速度。ShockBurst 模式通过允许与单片机低速通信而无线部分高速通信,减小了通信的平均消耗电流。

在 ShockBurstTM 接收模式下,当接收到有效的地址和数据时,IRQ 通知 MCU,随后 MCU 可将接收到的数据从 RX FIFO 寄存器中读出。

在 ShockBurstTM 发送模式下,nRF24L01 自动生成前导码及 CRC 校验,数据发送完毕后,IRQ 通知 MCU,减少了 MCU 的查询时间,也就意味着减少了 MCU 的工作量,同时减少了软件的开发时间。nRF24L01 内部有三个不同的 RX FIFO 寄存器(6 个通道共享此寄存器)和三个不同的 TX FIFO 寄存器。在掉电模式、待机模式和数据传输过程中,MCU 可以随时访问 FIFO 寄存器。这就允许 SPI 接口可以以低速进行数据传送,并且可以应用于 MCU 硬件上没有 SPI 接口的情况。

② 增强型 ShockBurstTM 模式。增强型 ShockBurstTM 模式可以使双向链接协议执行起来更为容易、有效。典型的双向链接为:发送方要求终端设备在接收到数据后有应答信号,以便于发送方检测有无数据丢失。一旦数据丢失,则通过重新发送功能将丢失的数据恢复。增强型 ShockBurstTM 模式可以同时控制应答及重发功能,而无需增加 MCU 的工作量。

2) 增强型 ShockBurstTM 发送过程

(1) 配置寄存器位 PRIM_RX 为低。

(2) 当 MCU 有数据要发送时,接收节点地址(TX_ADDR)和有效数据(TX_PLD)通过 SPI 接口写入 nRF24L01。发送数据的长度以字节计数,从 MCU 写入 TX_FIFO。当 CSN 为低

时,数据被不断地写入。发送端发送完数据后,将通道0设置为接收模式来接收应答信号,其接收地址(RX_ADDR_P0)与接收端地址(TX_ADDR)相同。

(3) 设置CE为高,启动发射。CE高电平持续时间最小为10 μ s。

(4) 无线系统上电、启动内部16 MHz时钟、无线发送数据打包和高速发送数据由MCU设定为1 Mb/s或2 Mb/s。

(5) 如果启动了自动应答模式(自动重发计数器不等于0, ENAA_P0=1),则无线芯片立即进入接收模式。如果在有效应答时间范围内收到应答信号,则认为数据成功发送到了接收端,此时状态寄存器的TX_DS位置高,并把数据从TX FIFO中清除掉;如果在设定时间范围内没有接收到应答信号,则重新发送数据。如果自动重发计数器ARC_CNT溢出超过了编程设定的值,则状态寄存器的MAX_RT位置高,不清除TX FIFO中的数据。当MAX_RT或TX_DS为高电平时,IRQ引脚产生中断。IRQ中断通过写状态寄存器来复位。如果重发次数在达到设定的最大重发次数时还没有收到应答信号,则在MAX_RX中断清除之前,不会重发数据包。数据包丢失计数器(PLOS_CNT)在每次产生MAX_RT中断后加1。也就是说,重发计数器ARC_CNT计算重发数据包次数,PLOS_CNT计算在达到最大允许重发次数时仍没有发送成功的数据包个数。

(6) 如果CE置低,则系统进入待机模式I;如果不设置CE为低,则系统会发送TX FIFO寄存器中下一包数据;如果TX FIFO寄存器为空并且CE为高,则系统进入待机模式II。

(7) 如果系统在待机模式II,则当CE置低后系统立即进入待机模式I。

3) 增强型 ShockBurstTM接收过程

(1) ShockBurstTM接收模式是通过设置寄存器中的PRIM_RX位为高来选择的。准备接收数据的通道必须被使能(EN_RXADDR寄存器),所有工作在增强型 ShockBurstTM模式下的数据通道的自动应答功能是由EN_AA寄存器来使能的,有效数据宽度是由RX_PW_Px寄存器来设置的。

(2) 接收模式由设置CE为高来启动。

(3) 130 μ s后nRF24L01开始检测空中信息。

(4) 接收到有效的数据包后(地址匹配,CRC检验正确),数据存储在RX FIFO中,同时RX_DR位置高,并产生中断。状态寄存器中RX_P_NO位显示数据是由哪个通道接收到的。

(5) 如果确认了信号,则发送确认信号。

(6) MCU设置CE脚为低,进入待机模式I(低功耗模式)。

(7) MCU将数据以合适的速率通过SPI口读出。

(8) 芯片准备好进入发送模式、接收模式或掉电模式。

4) 两种数据双向的通信方式

如果想要数据在双方向上通信,则PRIM_RX寄存器必须紧随芯片工作模式的变化而变化。处理器必须保证PTX和PRX端的同步性。在RX FIFO和TX FIFO寄存器中可能同时存有数据。

(1) 自动应答(RX)。自动应答功能减少了外部MCU的工作量,并且在鼠标/键盘等应用中也可以不要求硬件一定有SPI接口,因此降低了成本,减少了电流消耗。自动重应答功能可以通过SPI口对不同的数据通道分别进行配置。

在自动应答模式使能的情况下,收到有效的数据包后,系统将进入发送模式并发送确

认信号。发送完确认信号后,系统进入正常工作模式(工作模式由 PRIM_RX 位和 CE 引脚决定)。

(2) 自动重发功能(ART 或 TX)。自动重发功能针对自动应答系统的发送方。SETUP_RETR 寄存器用来设置启动重发数据的时间长度。在每次发送结束后,系统都会进入接收模式并在设定的时间范围内等待应答信号。接收到应答信号后,系统转入正常发送模式。如果 TX_FIFO 中没有待发送的数据且 CE 脚电平为低,则系统将进入待机模式 I。如果没有收到确认信号,则系统返回到发送模式并重发数据直到收到确认信号或重发次数超过设定值(达到最大的重发次数)。有新的数据发送或 PRIM_RX 寄存器配置改变时丢包计数器复位。

5) 数据包识别和 CRC 校验应用于增强型 ShockBurst™ 模式

每一包数据都包括两位的 PID(数据包识别)来识别接收的数据是新数据包还是重发的数据包。PID 识别可以防止接收端同一数据包多次送入 MCU。在发送方,每从 MCU 取得一包新数据后 PID 值加一。PID 和 CRC 校验应用在接收方,用来识别接收的数据是重发的数据包还是新数据包。如果在链接中有一些数据丢失了,则 PID 值与上一包数据的 PID 值相同。如果一包数据拥有与上一包数据相同的 PID 值,则 nRF24L01 将对两包数据的 CRC 值进行比较。如果 CRC 值也相同,则认为后面一包是前一包的重发数据包而被舍弃。

(1) 接收方。接收方将新接收数据包的 PID 值与上一包进行比较。如果 PID 值不同,则认为接收的数据包是新数据包。如果 PID 值与上一包相同,则新接收的数据包有可能与前一包相同。接收方必须确认 CRC 值是否相等。如果 CRC 值与前一包数据的 CRC 值相等,则认为是同一包数据并将其舍弃。

(2) 发送方。每发送一包新的数据,则发送方的 PID 值加一。CRC 校验的长度是通过 SPI 接口进行配置的。一定要注意 CRC 计算范围包括整个数据包:地址、PID 和有效数据等。若 CRC 校验错误,则不会接收数据包,这一点是接收数据包的附加要求。

6) 载波检测(CD)

当接收端检测到射频范围内的信号时,将 CD 置高,否则 CD 为低。内部的 CD 信号在写入寄存器之前是经过滤波的,内部 CD 高电平状态至少保持 128 μs 以上。

在增强型 ShockBurst™ 模式中,只有当发送模块没有成功发送数据时,推荐使用 CD 检测功能。当发送端 PLOS_CNT 显示数据包丢失率太高时,可检测 CD 值,如果 CD 为高(说明通道出现了拥挤现象),则需要更改通信频道。如果 CD 为低电平状态(距离超出通信范围),则可保持原有通信频道,但需进行其他调整。

7) 数据通道

nRF24L01 配置为接收模式时,可以接收 6 路地址不同但频率相同的数据。每个数据通道拥有自己的地址,并且可以通过寄存器来进行分别配置。

数据通道是通过寄存器 EN_RXADDR 来设置的,默认状态下只有数据通道 0 和数据通道 1 是开启状态的。

每一个数据通道的地址是通过寄存器 RX_ADDR_Px 来配置的,通常情况下不允许不同的数据通道设置完全相同的地址。

数据通道 0 有 40 位可配置地址,数据通道 1~5 的地址为 32 位共用地址+各自的地址(最低字节)。图 5-11 所示的是数据通道 1~5 的地址设置方法举例。所有数据通道可以设置为

多达 40 位，但是 1~5 数据通道的最低位必须不同。



图 5-11 通道 1~5 的地址设置

当从一个数据通道中接收到数据，并且此数据通道设置为应答方式时，nRF24L01 在收到数据后产生应答信号，此应答信号的目标地址为接收通道地址。

寄存器配置有些是针对所有数据通道的，有些则是针对个别的。如下设置(举例)是针对所有数据通道的。

- (1) CRC 使能/禁止；
- (2) CRC 计算；
- (3) 接收地址宽度；
- (4) 频道设置；
- (5) 无线数据通信速率；
- (6) LNA 增益；
- (7) 射频输出功率。
- (8) 寄存器配置

nRF24L01 的所有配置都是通过配置寄存器完成的。所有寄存器都是通过 SPI 口进行配置的。其配置指令如表 5-28 所示。

R_REGISTER 和 W_REGISTER 寄存器可能操作单字节或多字节寄存器。当访问多字节寄存器时，首先要读/写的是最低字节的高位。在所有多字节寄存器被写完之前，可以结束写 SPI 操作。在这种情况下，没有写完的高字节保持原有内容不变。例如，RX_ADDR_P0 寄存器的最低字节可以通过写一个字节给寄存器 RX_ADDR_P0 来改变。在 CSN 状态由高变低后可以通过 MISO 来读取状态寄存器的内容。

(1) SPI 接口。该器件的 SPI 接口是标准的 SPI 接口。其最大的数据传输率为 10 Mb/s，大多数寄存器是可读的。

(2) SPI 指令的设置。SPI 接口可能用到的指令将在下面进行说明。CSN 为低后，SPI 接口等待执行指令。每一条指令的执行都必须通过一次 CSN 由高到低的变化。

表 5-28 nRF24L01 指令集

指令名称	指令格式	含 义
R_REGISTER	000A AAAA	读配置寄存器。AAAAA是读操作的寄存器地址
W_REGISTER	001A AAAA	写配置寄存器。AAAAA是写操作的寄存器地址，只有在掉电模式和待机模式下可操作
R_RX_PAYLOAD	0110 0001	读RX有效数据，1~32字节。读操作全部从字节0开始。在读RX有效数据完成后，FIFO寄存器中的有效数据被清除(在接收模式下使用)
W_RX_PAYLOAD	1010 0000	写TX有效数据，1~32字节。写操作从字节0开始(在发射模式下使用)
FLUSH_TX	1110 0001	清除TX FIFO寄存器(应用于发射模式下)
FLUSH_RX	1110 0010	清除RX FIFO寄存器(应用于接收模式下)。在传输应答信号过程中不应执行此指令。也就是说，当传输应答信号过程中执行此指令时，将使得应答信号不能被完整地传输
REUSE_TX_PL	1110 0011	重新使用上一包有效数据。在CE为高的过程中，数据包被不断地重新发射。在发射数据包的过程中，必须禁止数据包重用功能
NOP	1111 1111	空操作，可以用来读状态寄存器

(3) SPI 指令的格式。

① 命令字：由高位到低位(每字节)。

② 数据字节：低字节到高字节，每一字节高位在前，如图 5-12 所示的时序。

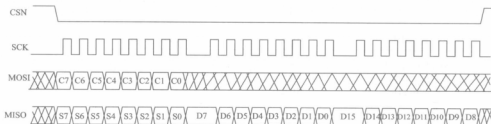


图 5-12 SPI 读操作时序

9) 中断

nRF24L01 的中断引脚(IRQ)为低电平触发，当状态寄存器中的 TX_DS、RX_DR 或 MAX_RT 为高时，触发中断。当 MCU 给中断源写“1”时，中断引脚被禁止。可屏蔽中断可以被 IRQ 中断屏蔽。默认状态下所有的中断源是被禁止的。

10) SPI 时序

nRF24L01 的 SPI 接口时序如图 5-12~图 5-14 所示。在写寄存器之前一定要进入待机模式或掉电模式。图 5-12~图 5-14 中，C7~C0 是 SPI 指令数据；S7~S0 是状态寄存器数据；D7~D0、D15~D8 是数据(低字节到高字节每个字节中高位在前)。

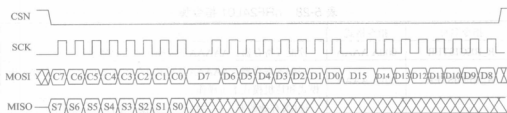


图 5-13 SPI 写操作时序

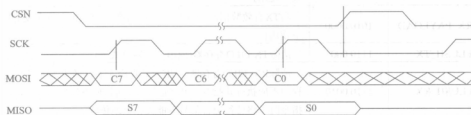


图 5-14 NOP 操作时序

4. 寄存器的地址与含义

nRF24L01 寄存器与其地址的对应关系见表 5-29。“地址位”的含义如表 5-30 所示。表 5-30 中，未定义的位在读出时值为 0。

表 5-29 寄存器与其地址的对应关系

寄存器名	地 址	寄存器名	地 址
配置寄存器	0x00	EN_AA Enhanced ShockBurst™	0x01
EN_RXADDR	0x02	SETUP_AW	0x03
SETUP_RETR	0x04	RF_CH	0x05
RF_SETUP	0x06	STATUS	0x07
OBSERVE_TX	0x08	CD	0x09
RX_ADDR_P0	0x0A	RX_ADDR_P1	0x0B
RX_ADDR_P2	0x0C	RX_ADDR_P3	0x0D
RX_ADDR_P4	0x0E	RX_ADDR_P5	0x0F
TX_ADDR	0x10	RX_PW_P0	0x11
RX_PW_P1	0x12	RX_PW_P2	0x13
RX_PW_P3	0x14	RX_PW_P4	0x15
RX_PW_P5	0x16	FIFO_STATUS	0x17
TX_PLD	N/A	RX_PLD	N/A

表 5-30 寄存器“地址位”的含义

地址	参 数	位	复位值	类型	描 述
00	Reserved	7	0	R/W	默认为 0
	MASK_RX_DR	6	0	R/W	可屏蔽中断(IRQ 脚)。“1”表示屏蔽,“0”表示允许
	MASK_TX_DS	5	0	R/W	可屏蔽中断(IRQ 脚)。“1”表示屏蔽,“0”表示允许
	MASK_MAX_RT	4	0	R/W	可屏蔽中断(IRQ 脚)。“1”表示屏蔽,“0”表示允许
	EN_CRC	3	1	R/W	CRC 使能。如果 EN_AA 为高,则 EN_CRC 强迫为高
	CRCO	2	0	R/W	CRC 模式。0~8 位 CRC 校验,1~16 位 CRC 校验
	PWR_UP	1	0	R/W	上电为“1”,掉电为“0”
01	PRIM_RX	0	0	R/W	1 为接收模式,0 为发射模式
	Reserved	7, 6	0	R/W	默认为 0
	ENAA_P5	5	1	R/W	数据通道 5 自动应答允许
	ENAA_P4	4	1	R/W	数据通道 4 自动应答允许
	ENAA_P3	3	1	R/W	数据通道 3 自动应答允许
	ENAA_P2	2	1	R/W	数据通道 2 自动应答允许
	ENAA_P1	1	1	R/W	数据通道 1 自动应答允许
02	ENAA_P0	0	1	R/W	数据通道 0 自动应答允许
	Reserved	7, 6	00	R/W	默认为 00
	ERX_P5	5	0	R/W	接收数据通道 5 允许
	ERX_P4	4	0	R/W	接收数据通道 4 允许
	ERX_P3	3	0	R/W	接收数据通道 3 允许
	ERX_P2	2	0	R/W	接收数据通道 2 允许
	ERX_P1	1	1	R/W	接收数据通道 1 允许
03	ERX_P0	0	1	R/W	接收数据通道 0 允许
	Reserved	7-2	0	R/W	默认为 0
04	AW	1-0	11	R/W	接收/发射地址宽度。01~11 表示 3~5 字节
	ARD	7-4	0000	R/W	自动重发延时。0000~1111 表示 336~4086 μ s
05	ARC	3-0	0011	R/W	自动重发计数。0000~1111 表示 1~15 次
	Reserved	7	0	R/W	默认为 0
05	RF_CH	6-0	0000010	R/W	设置 nRF24L01 工作通道频率

续表

地址	参 数	位	复位值	类型	描 述
06	Reserved	7-5	0	R/W	默认为 0
	PLL_LOCK	4	0	R/W	PLL_LOCK 允许, 仅应用于测试模式
	RF_DR	3	1	R/W	数据传输率。“0”为 1 Mb/s, “1”为 2 Mb/s
	RF_PWR	2-1	11	R/W	发射功率。00~11 表示-18~0 dBm
	LNA_HCURR	0	1	R/W	低噪声放大器增益
07	Reserved	7	0	R/W	默认为 0
	RX_DR	6	0	R/W	接收中断。收到有效数据置 1, 写 1 清除中断
	TX_DS	5	0	R/W	数据发送完成中断。写 1 清除中断
	MAX_RT	4		R/W	达到最多重复次数中断。写 1 清除中断
	RX_P_NO	3-1	0	R	接收数据通道号, 000~101 数据通道号
	TX_FULL	0	0	R	TX FIFO 寄存器满标志。“1”表示寄存器满
08	PLOS_CNT	7-4	0	R	数据包丢失计数器。发送新数据时此寄存器复位
	ARC_CNT	3-0	0	R	重发计数器发送新数据包时此寄存器复位
09	Reserved	7-1	0	R	默认为 0
	CD	0	0	R	载波检测
0A	RX_ADDR_P0	39-0	E7...E7	R/W	数据通道 0 接收地址
0B	RX_ADDR_P1	39-0	C2...C2	R/W	数据通道 1 接收地址
0C	RX_ADDR_P2	7-0	C3	R/W	数据通道 2 接收地址
0D	RX_ADDR_P3	7-0	C4	R/W	数据通道 3 接收地址
0E	RX_ADDR_P4	7-0	C5	R/W	数据通道 4 接收地址
0F	RX_ADDR_P5	7-0	C6	R/W	数据通道 5 接收地址
10	TX_ADDR	39-0	E7...E7	R/W	发送地址(先写低字节)
11	Reserved	7-6	0	R/W	默认为 0
	RX_PW_P0	5-0	0	R/W	接收数据通道 0 有效数据宽度(1~32 字节)
12	Reserved	7-6	0	R/W	默认为 0
	RX_PW_P1	5-0	0	R/W	接收数据通道 1 有效数据宽度(1~32 字节)
13	Reserved	7-6	0	R/W	默认为 0
	RX_PW_P2	5-0	0	R/W	接收数据通道 2 有效数据宽度(1~32 字节)
14	Reserved	7-6	0	R/W	默认为 0
	RX_PW_P3	5-0	0	R/W	接收数据通道 3 有效数据宽度(1~32 字节)
15	Reserved	7-6	0	R/W	默认为 0
	RX_PW_P4	5-0	0	R/W	接收数据通道 4 有效数据宽度(1~32 字节)

续表

地址	参 数	位	复位值	类型	描 述
16	Reserved	7-6	0	R/W	默认为 0
	RX_PW_P5	5-0	0	R/W	接收数据通道 5 有效数据宽度(1~32 字节)
17	Reserved	7	0	R/W	默认为 0
	TX_REUSE	6	0	R	若 TX_REUSE=1, 则当 CE=1 时, 发送上一数据包
	TX_FULL	5	0	R	TX FIFO 寄存器满标志。1 为寄存器满
	TX_EMPTY	4	1	R	TX FIFO 寄存器空标志。1 为寄存器空
	Reserved	3-2	0	R/W	默认为 0
	RX_FULL	1	0	R	RX FIFO 寄存器满标志。1 为寄存器满
	RX_EMPTY	0	1	R	RX FIFO 寄存器空标志。1 为寄存器空
N/A	TX_PLD	FF-0		W	—
N/A	RX_PLD	FF-0		R	—

5. nRF24L01 与 nRF24xx 的通信

nRF24L01 与 nRF24xx 在使用上是兼容的。

nRF24L01 从 nRF2401/nRF2402/nRF24E1/nRF24E2 接收数据要按以下方法进行：

- (1) 使用与 nRF2401/nRF2402/nRF24E1/nRF24E2 相同的 CRC 配置。
- (2) 设置 PRIM_RX 位为 1。
- (3) 相应通道禁止自动应答功能。
- (4) 与发射模块使用相同的地址宽度。
- (5) 与发射模块使用相同的频道。
- (6) 在 nRF24L01 和 nRF2401/nRF2402/nRF24E1/nRF24E2 两端都选择 1 Mb/s 的数据传输率。
- (7) 设置正确的数据宽度。
- (8) 设置 PWR_UP 和 CE 为高。

nRF24L01 发射, nRF2401/nRF2402/nRF24E1/nRF24E2 接收数据要按以下方法进行配置。

- (1) 使用与 nRF2401/nRF2402/nRF24E1/nRF24E2 相同的 CRC 配置。
- (2) 设置 PRIM_RX 位为 0。
- (3) 设置自动重发计数器为 0, 禁止自动重发功能。
- (4) 与 nRF2401/nRF2402/nRF24E1/nRF24E2 使用相同的地址宽度。
- (5) 与 nRF2401/nRF2402/nRF24E1/nRF24E2 使用相同的频道。
- (6) 在 nRF24L01 和 nRF2401/nRF2402/nRF24E1/nRF24E2 两端都选择 1 Mb/s 的数据传输率。
- (7) 设置 PWR_UP 为高。

(8) 发送与 nRF2401/nRF2402/nRF24E1/nRF24E2 寄存器配置数据宽度相同的数据长度。

(9) 设置 CE 为高, 启动发射。

6. 数据打包方法

数据打包的形式如表 5-31 所示。数据包的形式如表 5-32 所示。

表 5-31 数据打包的形式

增强型 ShockBurst™ 模式下的数据包形式				
前导码	地址(3~5 字节)	9 位(标志位)	数据(1~32 字节)	CRC 校验(0/1/2 字节)
ShockBurst™ 模式下与 nRF24xx 相兼容的数据包形式				
前导码	地址(3~5 字节)	数据(1~32 字节)	CRC 校验(0/1/2 字节)	

表 5-32 数据包的形式

前导码	前导码用来检测 0 和 1。芯片在接收模式下去除前导码, 在发送模式下加入前导码
地址	① 地址内容为接收机地址 ② 地址宽度可以是 3、4 或 5 个字节 ③ 地址可以对接收通道及发送通道分别进行配置 ④ 从接收的数据包中自动去除地址
标志位	① PID 为数据包识别。其中两位用来在接收到新的数据包后加 1 ② 七位保留用作将来与其他产品相兼容 ③ 当 nRF24L01 与 nRF2401/nRF24E1 通信时不起作用
数据	1~32 字节宽度
CRC	① CRC 校验是可选的 ② 0~2 字节宽度的 CRC 校验 ③ 8 位 CRC 校验的多项式是 X^8+X^2+X+1 ④ 16 位 CRC 校验的多项式是 $X^{16}+X^{12}+X^5+1$

5.3.2 器件的使用要点

1. 外围 RF 信息

1) 天线输出

ANT1 和 ANT2 输出脚给天线提供稳定的 RF 输出。这两个脚必须连接到 VDD 的直流通路, 或者通过 RF 扼流圈, 或者通过天线双极的中心点。在输出功率最大时(0 dBm), 推荐使用负载阻抗为 $15 + j88 \Omega$, 通过简单的网络匹配可以获得较低的阻抗(例如 50Ω)。

2) 输出功率调节

nRF24L01 的输出功率可调节, 如表 5-33 所示。

表 5-33 nRF24L01 的输出功率

RF_PWR	输出功率/dBm	电流消耗/mA
11	0	11.3
10	-6	9.0
01	-12	7.5
00	-18	7.0
工作条件: $V_{dd} = 3.0\text{ V}$, $V_{ss} = 0\text{ V}$, 环境温度为 27°C , 负载为 $15+j88\ \Omega$		

3) 晶振规格

晶振频率的精确度取决于出厂时精度的设置和在温度变化及老化过程中的稳定性。晶振规格如表 5-34 所示。

表 5-34 晶 振 规 格

频 率	CL	ESR	CO_max	精 度
16 MHz	$8\sim 16\text{ pF}$	$100\ \Omega$	7.0 pF	$\pm 60 \times 10^{-6}$

为了实现晶体振荡器低功耗和快速启动的目的, 建议使用表中容值较小的电容。最好采用晶振的并联等效电容 $Co = 1.5\text{ pF}$, 但考虑成本因素通常以 $Co_{\max} = 7.0\text{ pF}$ 代替 $Co = 1.5\text{ pF}$ 。

4) nRF24L01 与控制器共用晶振

当控制器驱动晶振给 nRF24L01 提供晶振输入(XC1)时, 电容 CL 只能通过控制器设置晶振精度为 60×10^{-6} 。所有输入信号的幅值都不能超过规定电压, 但任何内部的直流电压都可以超过。如果超过规定电压, 则将激发 ESD 结构, 会影响无线收/发效果。

当使用外部时钟时, XC2 引脚可以悬空。

2. PCB 板面设计及去耦

一个好的 PCB 布线对射频性能有很大影响。一个差的 PCB 板设计可能导致丢包, 甚至可能导致不能实现其应有的功能。

推荐使用至少两层板(包括一个地层)。nRF24L01 的直流供电电源应尽可能靠近芯片的 Vdd 引脚, 并且经高质量的 RF 电容去耦。最好用一个大电容(例如 $4.7\ \mu\text{F}$ 钽电容)并联一个小电容。nRF24L01 的供电电源必须经过很好的滤波并且与数字供电电源分离开来。

PCB 板应避免使用长的电源走线, 所有元器件的地及 Vdd 与去耦电容应尽可能地靠近 nRF24L01 芯片。如果在 PCB 板的顶层有铺铜“地”网, 则 Vss 应直接与铺铜面连接。如果在 PCB 板的底层有铺铜“地”网, 则应该在离 Vss 脚尽可能近的地方放置过孔连接。每个 Vss 至少应有一个过孔。

所有数字信号线和控制信号线都不能离晶振和电源线太近。

5.3.3 应用电路与编程

nRF24L01 与 STC12C5410 单片机组成的无线收发电路如图 5-15 所示。由于 STC12C5410 具有 SPI 接口, 因此可以直接与发射模块数字连接。考虑程序的通用性, 这里使用一般 I/O 口操作方法实现有关 nRF24L01 的编程。

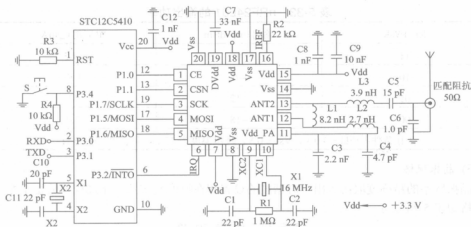


图 5-15 由 nRF24L01 构成的无线收发电路

```

#include <reg52.h>

sbit CE = P1^0;          /*RX 或 TX 模式选择端*/
sbit CSN = P1^1;         /*SPI 片选信号*/
sbit MDO = P1^5;         /*定义数据的输出脚*/
sbit MDI = P1^6;         /*定义数据的输入脚*/
sbit SCK = P1^7;         /*定义 SPI 总线的时钟*/
sbit IRQ = P3^2;         /*定义中断引脚*/
typedef unsigned char BYTE; /*BYTE 类型定义*/
#define uchar unsigned char /*定义状态字节的位变量*/

uchar bdata sta;

sbit RX_DR = sta^6;
sbit TX_DS = sta^5;
sbit MAX_RT = sta^4;

/* 定义发送地址 */
uchar const TX_ADDRESS[] = {0x34,0x43,0x10,0x10,0x01};

uchar rx_buf[20], tx_buf[20]; /*发送和接收的缓存区*/

uchar SPI_RW(uchar byte) /*SPI 读/写数据函数*/
{
    uchar bit_ctr;
    for(bit_ctr=0; bit_ctr<8; bit_ctr++) /*输出 8 个位*/
    {
        MDO = (byte & 0x80); /*输出数据, 从最高位到最低位*/
        byte = (byte << 1);
        SCK = 1;
        byte != MDI; /*读入数据*/
        SCK = 0;
    }
}

```

```

    return(byte);          /*返回读入的数据*/
}

uchar SPI_RW_Reg(BYTE reg, BYTE value)    /*读器件的寄存器*/
{
    uchar status;
    CSN = 0;                      /*SPI 模式开始*/
    status = SPI_RW(reg);          /*选择寄存器*/
    SPI_RW(value);
    CSN = 1;
    return(status);               /*返回状态字*/
}

BYTE SPI_Read(BYTE reg)            /*从寄存器中读一个字节*/
{
    BYTE reg_val;
    CSN = 0;      SPI_RW(reg);      /*选择要读的寄存器*/
    reg_val = SPI_RW(0);             /*读出数据*/
    CSN = 1;      return(reg_val);    /*返回值*/
}

/*写数据到 24L01 寄存器中, 用于 TX payload, RX/TX address*/
uchar SPI_Write_Buf(BYTE reg, BYTE *pBuf, BYTE bytes)
{
    uchar status, byte_ctr;
    CSN = 0;      status = SPI_RW(reg); /*选择要写的寄存器*/
    for(byte_ctr=0; byte_ctr<bytes; byte_ctr++)
        SPI_RW(*pBuf++);
    CSN = 1;      return(status);       /*返回状态字*/
}

/*从 24L01 寄存器中读数据, 用于 RX payload, RX/TX address*/
uchar SPI_Read_Buf(BYTE reg, BYTE *pBuf, BYTE bytes)
{
    uchar status, byte_ctr;
    CSN = 0;      status = SPI_RW(reg);
    for(byte_ctr=0; byte_ctr<bytes; byte_ctr++)
        pBuf[byte_ctr] = SPI_RW(0);    /*读出数据*/
    CSN = 1;      return(status);
}

void TX_Mode(void)                /*发送模式*/
{
    CE=0;
    SPI_Write_Buf(0x20+0x10, TX_ADDRESS, 5);    /*写发送地址*/
    SPI_Write_Buf(0x20+0x0A, TX_ADDRESS, 5);    /*设置 0 通道接收地址*/
    SPI_RW_Reg(0x20+0x01, 0x01);                /*通道 0 自动应答*/
    SPI_RW_Reg(0x20+0x02, 0x01);                /*使能通道 0*/
    SPI_RW_Reg(0x20+0x04, 0x1a);                /*自动重传配置 500 μs + 86 μs, 重传 10 次*/
    SPI_RW_Reg(0x20+0x05, 40);                 /*设置 RF channel 40*/
    SPI_RW_Reg(0x20+0x06, 0x07);                /*发射功率为 0 dBm, 速率为 1 Mb/s, LNA 为开*/
}

```

```

/*设置 PWR_UP 位, 使能 CRC(2 byte), Prim:TX. MAX_RT & TX_DS 中断使能*/
SPI_RW_Reg(0x20+0x00, 0x0e);
CE=1;
}

void RX_Mode(void) /*接收模式*/
{
    CE=0;
    SPI_Write_Buf(0x20+0x0A, TX_ADDRESS, 5); /*设置 0 通道接收地址*/
    SPI_RW_Reg(0x20+0x01, 0x01); /*通道 0 自动应答*/
    SPI_RW_Reg(0x20+0x02, 0x01); /*使能通道 0*/
    SPI_RW_Reg(0x20+0x05, 40); /*设置 RF channel 40*/
    SPI_RW_Reg(0x20+0x11, 20); /*接收有效数据宽度*/
    SPI_RW_Reg(0x20+0x06, 0x07); /*发射功率为 0 dBm, 速率为 1 Mb/s, LNA 为开*/
    /*设置 PWR_UP 位, 使能 CRC(2 byte), Prim:RX. RX_DR 中断使能*/
    SPI_RW_Reg(0x20+0x00, 0x0f);
    CE = 1; /*使能接收*/
}

void Send_Data(void) /*发送数据*/
{
    CE=0;
    SPI_Write_Buf(0xA0, tx_buf, TX_PLOAD_WIDTH); /*写入数据*/
    SPI_RW_Reg(0x20+0x07, SPI_Read(0x00+0x07)); /*清除 TX_DS 标志*/
    CE=1;
    Delay200ms(); /*延时 200 ms*/
}

void ISR_int0(void) interrupt 0 /*中断函数, 由于接收数据*/
{
    sta=SPI_Read(STATUS); /*读状态寄存器的值*/
    if(RX_DR) /*如果有(RX_DR)中断*/
        SPI_Read_Buf(RD_RX_PLOAD, rx_buf, TX_PLOAD_WIDTH); /*读 RX_FIFO 缓存区*/
    if(MAX_RT) /*如果收到数据满*/
        SPI_RW_Reg(FLUSH_TX, 0);
    SPI_RW_Reg(0x20+0x07, sta); /*清除 RX_DR 或 TX_DS 或 MAX_RT 中断标志*/
}

void main(void) /*测试函数*/
{
    uchar i=0;
    CE=0; CSN=1; SCK=0; /*初始化管脚*/
    TX_Mode(); /*设置发送模式*/
    for(i=0; i<20; i++)
        tx_buf[i]=i;
    Send_Data(); /*发送数据*/
    while(1); /*等待*/
}

```

5.4 新一代高性能 GPS 模块

5.4.1 总体描述

GPS 是全球卫星定位系统(Global Positioning System)的英文缩写。GPS 利用导航卫星进行测时和测距,以构成全球定位系统,能提供全天候的定位、授时、测速功能。

GPS 已被广泛应用于航天、航空、航海、运输、测量、勘探等诸多领域。随着数字大规模集成电路的发展和定位功能的需求, GPS 已经开始更多地嵌入到移动手持设备、消费电子产品中,如汽车电子、掌上电脑、蜂窝电话、数码相机、便携式 DVD 播放器、资产管理等领域。

1. GPS 的基本原理

GPS 是一种精密的卫星导航系统。该系统由 24 颗绕地球旋转的卫星组成,卫星连续不断地发送位置和时间信息。这些卫星均匀地分布在 6 个轨道上,每个轨道有 4 颗卫星。地面 GPS 接收机可接收 5~12 颗卫星信号。为实现地面定位功能, GPS 接收机至少需要接收 4 个卫星信号,其中 3 个信号用来计算 GPS 接收机的纬度、经度和海拔高度,第 4 个信号提供同步时间校准。

GPS 系统主要分为三个部分:其一是卫星,在天上提供定位信息;其二是控制系统,在地面维护卫星的正常运转,保证卫星的健康状态;其三是接收机,即 GPS 模块或一般用户所使用的部分。其定位原理是将地球分为 12 个横切面,每个横切面上有两颗定位卫星,互成 180° 的夹角,因而站在地球上的任一点,头顶上总有 12 颗定位卫星。卫星与卫星之间的距离、坐标和角度是已知的,卫星和人之间的距离是可测量的。根据几何原理,通常接收机只要接到 3 颗卫星的信息便可确定二维坐标即经纬度,接收到四颗卫星的信息便可确定海拔高度。

目前卫星(多达 27 颗)网络运行于非同步、近地轨道并覆盖全球,保证了定位系统的运行,而 GPS 接收机至少需要锁定 4 颗卫星,才能提供定位信息。这些卫星广播或发送的长系列码(或数字组合)称为伪随机码。GPS 接收机通过已知的卫星伪随机码、光速以及保持卫星位置的查询表等参数,就能够计算出卫星的传输时间,再将传输时间转为距离。在多个卫星(大于 4)的条件下,通过求解三角方程就可以算出 GPS 接收机的位置,即提供了用户的位置。

2. GPS 的组成框图

GPS 芯片是由一块射频集成电路、一块数字信号处理电路和标准嵌入式 GPS 软件构成的。射频集成电路用于检测和处理 GPS 射频信号;数字信号处理电路用于处理中频信号;标准嵌入式 GPS 软件用于搜索和跟踪 GPS 卫星信号,并根据这些信号求解用户坐标和速度,一般是无线手持设备、汽车、便携式计算设备以及一些 CPS 专业用户。图 5-16 所示为手持式 GPS 系统结构框图。

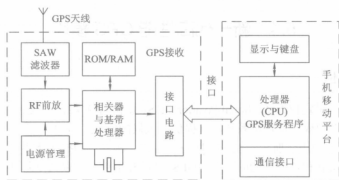


图 5-16 手持式 GPS 系统结构框图

3. 新一代 GPS 的特点

以三代 GPS 为代表的主要芯片(模块)具有以下特征:

- (1) 定位时间快。无论冷启动、温启、热启,重捕时间均快 5~30 s(与二代相比)。
- (2) 高感度。在高楼、树荫、桥下、遮挡、隧洞、窗口、车内,甚至车底盘下仍可很快定位 4 颗以上卫星。一般无遮挡天空就可定位。
- (3) 抗干扰性能强。高压线、电场、磁场、高速动态、微波、手机、同频干扰的环境下仍能正常工作。
- (4) 功耗低。新一代 GPS 降低了功耗,甚至有睡眠状态(静态不工作),可以节电,提高了产品的待机时间。
- (5) 性价比高。新一代 GPS 体积小,重量轻,这是社会的需求和发展趋势,可以扩大更多的应用范围和领域。实际上三代芯片的价格与二代相同,甚至更低,但功能提高很多。
- (6) 硬件/软件融合。整个参考设计基于 GPS 接收器前端,软件获取 GPS 前端接收器输出和卫星定位信息,然后下载卫星星历,利用新技术进行信号处理,提取卫星信号,产生有效的定位信息。

5.4.2 GPS 模块介绍

GPS 模块(OEM 板)也就是非独立 GPS 接收器,即模块只能提供 NMEA 协议标准定位信息给输出接口(或机身设备),并最终由计算终端将数据换算为地点坐标。目前市场上的 GPS 模块较多。该类模块又可以分为 UART、蓝牙、CF、USB 等几种类型。通常 GPS 模块体积小,便于携带。

目前国际上较流行的 GPS 模块(OEM 板或芯片)主要有以下几种。

1. MT3318 蓝牙 GPS 接收器

MT3318 芯片组同时整合了 GPS 模块与蓝牙模块,采用了单芯片解决方案, GPS 接收、解码、蓝牙等系统都采用单一芯片解决,具有功能强、精度高、功耗低和体积小特点。

MT3318 芯片的最大优势就是对“漂移”处理要优于其他模块,具备快速定位及追踪 32 颗卫星的能力,可快速定位及重新定位,而且灵敏度同样达到了惊人的“-159 dBm”。

该器件采用小型 16 脚表贴封装(11.5 mm × 13 mm × 1.9 mm)。

1) MT3318 的性能指标

- (1) 32 并列卫星追踪频道, 频率为 L1 频段 1575.42 MHz。
- (2) 接收 C/A 码。
- (3) 冷开机时间<36 s, 暖开机时间<33 s, 热开机时间<1 s。
- (4) 重新定位时间<0.1 s。
- (5) 接收器准确度一般小于 3 m, 定位精度<2.2 m。
- (6) 灵敏度(典型值)为-159 dBm。
- (7) 电源电压为 1.8~6.0 V, 工作温度为-30~85℃。
- (8) 功耗在 2.8 V 时, 为 30~55 mA。

2) 输出格式

- (1) 数据输出速率: 9600/38400/57600/115200 b/s(默认为 9600)。
- (2) UART 串行数据位: 1 个起始位、8 个数据位和 1 个停止位(共 10 位)。
- (3) 数据格式: GGA(1 s)、GSA(1 s)、RMC(1 s)、VTG(1 s)、GSV(5 s)。

2. SiRFstar III GPS 接收器

SiRFstar III 是最新的第三代 GPS 接收器。SiRFstar III(MG-S02)使民用 GPS 芯片在性能方面又上了一个高度, 灵敏度比以前的产品大为提升。这一芯片通过采用 20 万次/频率的相关器(Correlators)提高了灵敏度, 冷开机、暖开机、热开机的时间分别达到 42 s、38 s、8 s, 可同时追踪 20 个卫星信道。

SiRFstar III 具有超强的噪声滤除能力, 特别在城区或峡谷地域这种性能更能表现出来。一般来说, GPS 模块在接收到空中信号的同时, 也会收到坚硬表面(比如悬崖、高楼)反射的信号, 也就是通信中常说的多径衰落。事实上 GPS 是通过计算卫星信号到达接收器所用的时间来确定用户的所在位置的, 所以那些反射信号就成为难以摆脱的干扰源。不过 SiRFstar III 在强大的数字处理能力下, 通过众多的判断算法会剔除掉无用的干扰, 并且把那些被其他 GPS 产品所忽略的较弱的有用信号筛选出来。另外, SiRFstar III 具备非常快的收集信号的能力, 当 GPS 掉线以后它也能够快速重新定位。

该器件采用小型 24 脚表贴封装(20 mm × 19.3 mm × 2.6 mm)。

1) SiRFstar III 的性能指标

- (1) 20 并列卫星追踪频道, 频率为 L1 频段 1575.42 MHz。
- (2) 接收 C/A 码(芯片速率为 1.023 MHz)。
- (3) 冷开机时间<42 s, 暖开机时间<38 s, 热开机时间<8 s。
- (4) 重新定位时间<0.1 s(平均值)。
- (5) 接收器准确度, 一般小于 10 m, 定位精度<5 m。
- (6) GPS 同步时间为 1 ms, 灵敏度(典型值)为-159 dBm。
- (7) 电源电压为 3.3 V±10%, 工作温度为-40~85℃;
- (8) 功耗在 3.3 V 时(典型值)为 75 mA。

2) 输出格式

- (1) 数据输出速率: 9600 b/s(默认)。

- (2) UART 串行数据位: 1 个起始位、8 个数据位和 1 个停止位(共 10 位)。
- (3) 数据格式: GGA(1 s)、GSA(1 s)、GSV(5 s)、RMC(1 s)、VTG(1 s)。

3. XT55 多功能 GPRS/GPS 模块

XT55 是一款集合了三频 GSM/GPRS 和 GPS 卫星导航接收器的模块,也可以把它叫做二合一模块。这种组合提高了整体性能,降低了系统的成本和尺寸。这两项技术的结合实现了对货物、车辆和人员的无间断追踪。这种将 GSM/GPRS 以及 GPS 功能集于一身的新型模块在车辆跟踪、导航、紧急呼叫和保安领域将会得到广泛应用。

XT55 采用 80 脚引线,紧凑的模块体积为 $35\text{ mm} \times 53\text{ mm} \times 5.1\text{ mm}$ 。

1) 主要功能

- (1) 集成有 GSM/GPRS 三频(900/1800/1900 MHz)通信电路。
- (2) 集成有 TCP/IP 协议。
- (3) 集成有新一代 GPS 接收电路。

2) GSM/GPRS 的主要性能

- (1) 输出功率: 900 MHz 时为 2 W, 1800 MHz 时为 1 W, 1900 MHz 时为 1 W。
- (2) GSM 通信: 采用标准的 AT 命令(同 GSM07.07 和 07.05 版本)。
- (3) GPRS 通信: 点到点、字符和 PDU 通信模式。
- (4) 语音: 符合标准的手机语音通话规范。
- (5) 数据传输: 下载速率为 85.6 kb/s。
- (6) 电源电压: 3.3~4.8 V。

3) GPS 的主要性能

- (1) 12 个卫星追踪通道, 频率为 L1 频段 1575.42 MHz。
- (2) 接收 C/A 码(芯片速率 1.023 MHz)。
- (3) 冷开机时间<48 s, 暖开机时间<38 s, 热开机时间<8 s。
- (4) 接收器准确度一般小于 15 m, 定位精度<8 m。
- (5) GPS 同步时间为 1 μs 。
- (6) 电源电压为 3.3 V + 10%, 功耗为 200 mW(在 3.3 V 时)。

4) 接口

- (1) 外接 SIM 卡。
- (2) 具有数字和模拟音频接口。
- (3) GSM 串行接口一个, GPS 串行接口两个。
- (4) 两个不同的天线接口。

4. LEA-5M/Q/H GPS 模块

LEA-5M 和 LEA-5Q 是单机式 GPS 模块,能够以更轻巧的 $17\text{ mm} \times 22\text{ mm} \times 2.4\text{ mm}$ 封装尺寸提供前一代 LEA-4 产品所具备的全部特性与连接性。LEA-5 M/Q 采用以只读存储器(ROM)为基础的架构设计,无需昂贵的外部 Flash EPROM,因此非常适合受限于严格成本考虑的大量产品应用。

以 u-blox 的第五代定位引擎 u-blox 5 为基础,此模块的效能优异,利用其具备 50 个通道的搜寻引擎以及 100 万个以上相关器(Correlators),能够同步追踪 GPS 和伽利略(GALILEO)的卫星信号,并在 1 s 内完成卫星信号的定位。

LEA-5M GPS 模块具有-160 dBm 的灵敏度,即使在商场与都会区等无法收到直接卫星信号或信号微弱的环境也能进行追踪与定位。LEA-5Q GPS 模块则拥有更多的界面选项,并增加了 u-blox 新的 KickStar 微弱信号快速获取技术,能快速实现定位。

1) LEA-5M/Q/H 的性能指标

(1) 内建 50 个通道,包括 32 通道的捕获引擎和 18 通道的跟踪引擎,捕获引擎包括 100 万个相关器。

(2) 掩膜有导航算法,无需外围或内嵌 Flash,进一步简化了系统结构,节约了成本。

(3) 全独立工作方式,直接输出 NMEA-0183 数据,不占用系统任何资源,可配合任何软件或硬件平台,更新维护成本低。

(4) 捕获与跟踪灵敏度为-160 dBm,冷启动灵敏度为-145 dBm。

(5) 冷开机时间<29 s,暖开机时间<27 s,热开机时间<1 s。

(6) 定位精度:2.5 m CEP, DGPS/SBAS 2 m CEP。

(7) 定位数据更新率最高达到 4 Hz,为高速移动提供精确的定位。

(8) 硬件兼容伽利略系统,并支持 WAAS、EGNOS、MSAS、GAGAN 等 SBAS 系统。

(9) 宽电源电压范围,支持 1.8 V 和 2.5~4.8 V,工作温度为-30~85℃。

2) 接口

(1) 一个 USB 口(V2.0 全速 12 Mb/s)。

(2) 一个 UART 通信串口(1 个起始位、8 个数据位和 1 个停止位)。

(3) 一个 SPI 接口。

5. STA2051 32 位系统芯片 GPS 基带控制器

STA2051 在同一个芯片上集成了 32 位 ARM7TDMI 微控制器、12 通道 GPS 相关器、2 Mb 闪存、64 KB RAM 以及包括模/数转换器、CAN 和 USB 接口在内的全套外设,因此,该芯片是目前灵活性最高的产品之一。

STA2051 片内所增加的处理软件可在卫星能见度不高的地区跟踪卫星,当重新进入卫星可见度高的地区时,具有很高的卫星数据采集灵敏度。此外,还提供“航位推测”定位功能,当无法接收 GPS 卫星信号或信号较弱时,“航位推测”定位功能可以利用车载传感器执行定位任务。例如在隧道和深谷中,汽车可利用里程表和角度传感器(陀螺仪或 ABS 传感器)分别将车速及前后运动数据和方向指示数据等信息馈入 STA2051 的输入端,实现地理再现定位。

1) STA2051 的主要特性

(1) 12 通道 GPS 硬件相关器,ARM7TDMI 32 位 RISC 引擎。

(2) 内嵌 256 KB + 16 KB Flash,64 KB SRAM。

(3) EMI 扩展存储器接口(仅 TQFP144),分为 4 区,最大可扩展至 64 MB。

(4) 独立的 RTC(Real Time Clock)带唤醒功能。

(5) CAN 控制器(高达 1 Mb/s)、四路 UARTS、双 SPI、双 I²C 和 USB 等串口通信。

(6) 四路 16 位多功能定时器(捕捉、比较、计数、PWM)。

(7) 看门狗及唤醒控制(RTC 或外部触发)和 HDLC 控制。

(8) 4 通道 12 位 ADC 变换器。

(9) 3.3 V 单电源供电,工作温度为-40~+85℃。

2) GPS 模块的设计方案

基于 STA2051 的功能特点:只要在前端增加一个射频接收电路(如 STB5610 集成芯片),就可构成完整的 GPS 接收模块。STA2051 内嵌闪存,这意味着可以通过 CAN 或 USB 接口给该 GPS 应用软件快速升级。加上高性能的 32 位 ARM7 具有丰富的硬件资源和处理能力,不仅可作为高性价比的 GPS 处理器,也可作为系统的其他设计用途,从而极大地降低了系统成本并简化了整个硬件电路。

6. E531 GPS 模块

E531 是 12 通道的 GPS 接收机模块,同时可以跟踪多达 12 颗 GPS 卫星,跟踪性能优越,从而能够快速定位。该 GPS 模块功耗小,数据更新率每秒一次,其优良的性能既能满足陆地导航的灵敏度需求,也能够满足飞行器的动态需求。

该 GPS 模块从硬件和软件上都十分易于使用,非常适合进行系统集成。数据通信通过 TTL 电平的串行口就可实现卫星参数和定位信息。

1) E531GPS 模块的性能指标

- (1) 12 并列卫星追踪频道,频率为 1575.42 ± 1.023 MHz。
- (2) 冷开机时间 <52 s, 暖开机时间 <35 s, 热开机时间 <8 s。
- (3) 自由定位 2 min。
- (4) 接收器准确度一般小于 12 m, 定位精度 <6 m(2DRMS)。
- (5) 接收机灵敏度为 -150 dBm(跟踪)、-137 dBm(捕获)。
- (6) 电源电压为 3.3~3.6 V, 工作温度为 -30~80℃。
- (7) 功耗为 52 mA(跟踪模式)、4 mA(睡眠模式)。

2) 接口

- (1) 接口特性为 TTL 电平输出,可选波特率为 4800、9600、19 200、57 600 和 115 200 b/s。
- (2) 串口 0 接口协议的默认波特率为 4800 b/s,串口 1 接口协议的默认波特率为 115 200 b/s。
- (3) 输出格式(默认)为 NMEA0183 版本 3.0 的 ASCII 码语句,包括 GGA、GSA、GSV、RMC,可选输出格式为 GLL、VTG、ZDA、DTM。

5.4.3 数据格式

GPS 模块的数据输出格式是以美国国家海洋电子协会(National Marine Electronics Association)的 NMEA 0183 ASCII 码接口协议为基础的。此协议是为了在不同的 GPS 导航设备中建立统一的 RTCM 标准而制定的,也是目前 GPS 接收机上使用最广泛的协议。大多数常见的 GPS 接收机、GPS 数据处理软件、导航软件都遵守或者至少兼容这个协议。

传输速率可自定义,缺省波特率为 4800 b/s。

1. NMEA 0183 输入语句

NMEA 0183 的输入语句用来完成模块参数的设置。所有的语句必须以 \$PFST 开头,以 <CR><LF> 结束,也就是 ASCII 字符“回车”(十六进制的 0D)和“换行”(十六进制的 0A)。最后的校验码*hh 是用作奇偶校验的数据。在通常使用时,它并不是必需的。但当周围环境中有较强的电磁干扰时,则推荐使用校验码。hh 代表了“\$”和“*”之间所有字符的按位异或值(不包括这两个字符)。

1) 海拔高度援助模式的设置

如果设置海拔高度援助模式有效,即用户可以自己输入一个预知的海拔高度,则可以在3颗卫星的情况下实现3D定位。海拔高度援助模式的默认值是无效的,即用内部的海拔高度。

海拔高度援助模式的设置格式是:

\$PFST, ALTAID, <n>, <x.x>

其中, <n>是援助模式, 0表示无效, 1表示有效; <x.x>是用户输入的海拔高度, 单位是 m。

例如, “\$PFST, ALTAID, 1, 34.2”表示输入的援助海拔高度为 34.2 m, “\$PFST, ALTAID, 0”表示用默认海拔高度。

2) 波特率的设置

波特率的设置格式是:

\$PFST, CONF, 21, <m>

其中, <m>是要设的波特率, 单位是 Hz/s。

例如, “\$PFST, CONF, 21, 9600”表示波特率设为 9600 b/s。

3) 接收机的初始化设置

接收机的初始化设置语句可以为 GPS 接收机提供初始化位置和时间信息, 从而帮助捕获 GPS 卫星。接收机收到这条语句后将重新开始搜索卫星。其格式为

\$PFST, INITAID, <time>, <date>, <lat>, <N/S>, <long>, <E/W>, <altitude>

其中, <time>是 UTC 时间, hhmmss.dd(时:分:秒); <date>是 UTC 日期, ddmmyy(日:月:年); <lat>是纬度, ddmm.mmmm(度:分); <N/S>是纬度 N(北半球)或 S(南半球); <long>是经度, dddmm.mmmm(度:分); <E/W>是经度 E(东经)或 W(西经); <altitude>是海拔高度(m)为 1~5 位数。

例如, “\$PFST, INITAID, 131500.78, 100102, 6016.3075, N, 02458.3817, E, 40”表示: 时间为 13:15:00.78 (UTC), 日期是 10 10-Jan Jan-2002, 纬度是 6016.3075, N 是北半球, 经度是 02458.3817, E 是东经, 海拔高度是 40 m。

又如, “\$PFST, INITAID, , 6016.3075, N, 02458.3817, E(只有位置)”中, 内容可省, 但逗号不能省。

4) 输出语句和波特率的更改

输出语句和波特率的更改设置格式是:

\$PFST, NE MA, <mAsk>, <speed>

其中, <mAsk>是输出格式的数字和, GSV 格式是 0x0001, GSA 格式是 0x0003, ZDA 格式是 0x0004, PPS 格式是 0x0010, FOM 格式是 0x0020, GLL 格式是 0x1000, GGA 格式是 0x2000, VTG 格式是 0x4000, RMC 格式是 0x8000; <speed>是波特率(200、2400、4800、9600、19 200、57 600、115 200 b/s 中的一个)。

例如, “\$PFST, NMEA, 7003, 4800”表示输出 GLL、GGA、VTG、GSA 和 GSV 格式, 波特率为 4800 b/s。

5) 恢复出厂输出语句和波特率的设置

格式如下:

\$PFST, RESTORE

6) 休眠模式的设置

如果在导航期间进入休眠模式,则唤醒后将继续导航;如果在空闲模式期间进入休眠模式,则唤醒后继续进入空闲模式。

休眠模式设置的格式是:

\$PFST,SLEEP,<ssss>,<mmmm>

其中,<ssss>是进入休眠模式后的定时唤醒时间,最大为 4 292 967 s,约 49 天 17 小时;
<mmmm>是唤醒源,如果省略,则将从 UART 口或 I/O 脚唤醒。

例如,“\$PFST, SLEEP, 900”表示 900 s(15 min)后由 UART 口唤醒。

2. NMEA 0183 输出语句

NMEA 0183 输出语句定义了 GPS 模块的数据输出格式。其输出一般有“GSV、GSA、ZDA、PPS、FOM、GLL、GGA、VTG 和 RMC”9 种格式的定位信息。常用的几种格式如下所述。

1) GPS 定位信息(GGA)

输出数据格式:

\$GPGGA, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, <10>, <11>,
<12>*hh<CR><LF>

其中,“<>”的含义如下:

<1>表示 UTC 时间: hhmmss.ddd(时分秒);

<2>表示纬度: ddm.ddd(度分)格式(前面的 0 也将被传输);

<3>表示纬度方向: N(北半球)或 S(南半球);

<4>表示经度: ddm.ddd(度分)格式(前面的 0 也将被传输);

<5>表示经度方向: E(东经)或 W(西经);

<6>表示 GPS 的状态: 0=未定位, 1=定位;

<7>表示正在使用的卫星数量: 00~12(前面的 0 也将被传输);

<8>表示水平精度因子: 0.5~99.9;

<9>表示海平面高度: -9999.9~99999.9;

<10>表示地球椭球面相对大地水准面的高度;

<11>差分时间: 从最近一次接收到差分信号开始的秒数,如果不是差分定位,则为空;

<12>表示差分站 ID 号: 000~123。

例如:

\$GPGGA, 084053.039, 6016.3051, N, 02458.3735, E, 0, 00, 0.0, 46.6, M,
18.2, M, , *5D

2) 当前卫星信息(GSA)

输出数据格式:

\$GPGSA, <1>, <2>, <3>, <3>, <3>, <3>, <3>, <3>, <3>, <3>, <3>,
<3>, <3>, <4>, <5>, <6>*hh<CR><LF>

其中,“<>”的含义如下:

<1>表示模式: M=手动, A=自动;

<2>表示定位类型: 1=没有定位, 2=2D 定位, 3=3D 定位;

<3>表示 PRN 码(伪随机噪声码),为正在用于解算位置的卫星号(01~32,前面的0也将被传输);

<4>表示 PDOP 位置精度因子:0.5~99.9;

<5>表示 HDOP 水平精度因子:0.5~99.9;

<6>表示 VDOP 垂直精度因子:0.5~99.9。

例如:

```
$GPGSA, A, 3, 06, 10, 15, 16, 21, 25, 30, , , , , 2.1, 1.2, 1.8*38
```

3) 可见卫星信息(GSV)

输出数据格式:

```
$GPGSV, <1>, <2>, <3>, <4>, <5>, <6>, <7>, ...<4>, <5>, <6>, <7>*hh<CR><LF>
```

其中,“<>”的含义如下:

<1>表示 GSV 语句的总数目;

<2>表示本句 GSV 语句的数目;

<3>表示可见卫星的总数(00~12,前面的0也将被传输);

<4>表示 PRN 码(伪随机噪声码)(01~32,前面的0也将被传输);

<5>表示卫星仰角(00~90°,前面的0也将被传输);

<6>表示卫星方位角(000~359°,前面的0也将被传输);

<7>表示载噪比 C/N₀(00~99dB/Hz,没有跟踪到卫星时为0,前面的0也将被传输)。

注意:<4>、<5>、<6>、<7>信息将按照每颗卫星进行循环显示,每条 GSV 语句最多可以显示4颗卫星的信息。其他卫星信息将在下一序列的 NMEA0183 语句中输出。

例如:

```
$GPGSV, 4, 1, 14, 03, 66, 207, 50, 08, 09, 322, 44, 11, 01, 266, 42,
14, 00, 155, 00*79
```

```
$GPGSV, 4, 2, 14, 15, 41, 088, 48, 17, 21, 083, 44, 18, 57, 087, 51,
21, 57, 173, 50*78
```

```
$GPGSV, 4, 3, 14, 22, 05, 203, 00, 23, 52, 074, 49, 26, 17, 028, 44,
27, 00, 300, 00*79
```

```
$GPGSV, 4, 4, 14, 28, 32, 243, 00, 31, 48, 286, 00*70
```

4) 推荐定位信息(RMC)

输出数据格式:

```
$GPRMC, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, <10>, <11>,
<12>*hh<CR><LF>
```

其中,“<>”的含义如下:

<1>表示 UTC 时间:hhmmss.ddd(时分秒)格式;

<2>表示定位状态: A=有效定位, V=无效定位;

<3>表示纬度: ddm m.mmmmm(度分)格式(前面的0也将被传输);

<4>表示纬度方向: N(北半球)或 S(南半球);

<5>表示经度: dddmm.mmmmm(度分)格式(前面的0也将被传输);

<6>表示经度方向: E(东经)或 W(西经);

- <7>表示地面速率: 000.0~999.9 节, 前面的 0 也将被传输;
- <8>表示地面航向: 000.0~359.9 度, 以真北为参考基准, 前面的 0 也将被传输;
- <9>表示 UTC 日期: ddmmyy(日月年)格式;
- <10>表示磁偏角: 000.0~180.0 度, 前面的 0 也将被传输;
- <11>表示磁偏角方向: E(东)或 W(西);
- <12>表示模式指示: A=自主定位, N=数据无效。

例如:

```
$GPRMC, 095035.091, A, 6016.3066, N, 02458.3832, E, 1.08, 210.6, 131204, 6.1, E, A*0A
```

5) 地面速度信息(VTG)

输出数据格式:

```
$GPVTG, <1>, T, <2>, M, <3>, N, <4>, K, <5>*hh<CR><LF>
```

其中, “<>”的含义如下:

- <1>表示以真北为参考基准的地面航向: 000~359°, 前面的 0 也将被传输;
- <2>表示以磁北为参考基准的地面航向: 000~359°, 前面的 0 也将被传输;
- <3>表示地面速率: 000.0~999.9 节, 前面的 0 也将被传输;
- <4>表示地面速率: 0000.0~1851.8 km/h, 前面的 0 也将被传输;
- <5>表示模式指示: A=自主定位, N=数据无效。

例如:

```
$GPVTG, 202.6, T, 208.7, M, 0.38, N, 0.7, K, A*0D
```

6) 定位地理信息(GLL)

输出数据格式:

```
$GPGLL, <1>, <2>, <3>, <4>, <5>, <6>, <7>*hh<CR><LF>
```

其中, “<>”的含义如下:

- <1>表示纬度: ddmm.mmmm(度分)格式(前面的 0 也将被传输);
- <2>表示纬度方向: N(北半球)或 S(南半球);
- <3>表示经度: dddmm.mmmm(度分)格式(前面的 0 也将被传输);
- <4>表示经度方向: E(东经)或 W(西经);
- <5>表示 UTC 时间: hhmmss.ddd(时分秒)格式;
- <6>表示定位状态: A=有效定位, V=无效定位;
- <7>表示模式指示: A=自主定位, N=数据无效。

例如:

```
$GPGLL, 6016.3073, N, 02458.3817, E, 090110.010, A, A*61
```

7) UTC 时间和日期信息(ZDA)

输出数据格式:

```
$GPZDA, <1>, <2>, <3>*hh<CR><LF>
```

其中, “<>”的含义如下:

- <1>表示 UTC 时间: hhmmss.ddd(时分秒)格式;
- <2>表示 UTC 日期: dd, mm, yyyy(日月年)格式;

<3>表示时区设置: xx, yy(分钟, 小时)。

例如:

\$GPZDA, 061724.046, 17, 04, 2003, 00, 00*61

5.4.4 应用电路与编程

目前, GPS 接收模块的应用非常广泛。图 5-17 是 GPS 在“汽车定位系统”中的典型应用。图中, GPS 模块用来接收“卫星定位信息”并将数据通过串口送入 C8051F340 单片机的 UART1 接口。单片机除了完成 GPS 信息的获取外, 还把“定位信息”及“车载信息”由 UART0 口通过手机通信模块发送到监控中心, 实现在相关地图上再现“地理位置”。

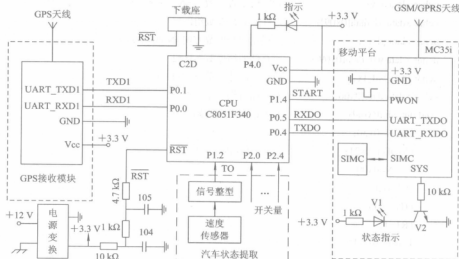


图 5-17 GPS 模块的应用

GPS 接收模块只要处于工作状态就会源源不断地把 GPS 定位信息(NE MA0183 语句)通过串口传送到单片机中。其发送到计算机的数据主要由帧头、帧尾和帧内数据组成, 根据数据帧的不同, 帧头也不相同, 主要有“\$GPGGA”、“\$GPGSA”、“\$GPGSV”以及“\$GPRMC”等。这些帧头标识了后续帧内数据的组成结构, 各帧均以回车符和换行符作为帧尾标识一帧的结束。对于“车载定位”, 所关心的定位数据如经纬度、速度、时间等均可以从“\$GPRMC”帧中获取得到, 即

\$GPRMC, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, <10>, <11>, <12>*hh<CR><LF>

“<>”中的含义见 NE MA0183 语句。

至于其他几种帧格式, 除了特殊用途外, 平时并不常用。虽然接收机也在源源不断地向主机发送各种数据帧, 但在处理时一般先通过对帧头的判断而只对“\$GPRMC”帧进行数据的提取处理。如果情况特殊, 则需要从其他帧获取数据, 处理方法与之也是完全类似的。由于帧内各数据段由逗号分割, 因此在处理缓存数据时一般通过搜寻 ASCII 码“\$”来判断是否为帧头, 在对帧头的类别进行识别后再通过对所经历逗号个数的计数来判断当前正在处理的是哪一种定位导航参数, 然后做出相应的处理。

下面是用 C51 所实现的“GPS 定位信息”的获取程序。

```
#include <reg52.h>
sbit sw1=P2^0;           /*定义开关量 1 引脚*/
sbit sw2=P2^1;           /*定义开关量 2 引脚*/
sbit start=P1^4;         /*定义 GPRS 启动引脚*/

#define uchar unsigned char
#define JS MAX 0x2000    /*定义超时量*/

uchar xdata x_data[80];  /*接收 GPS 数据区*/
uchar xdata time[0x06];  /*存储时间数组区*/
uchar idata a_v;         /*定位是否有效, =A 表示有效, =V 表示无效*/
uchar xdata jd_data[0x09]; /*存储的经度数组区*/
uchar idata jd_fx;       /*存储的经度数方向=N 或=S*/
uchar xdata wd_data[0x09]; /*存储的纬度数组区*/
uchar idata wd_fx;       /*存储的纬度数方向=E 或=W*/
uchar xdata date[0x06];  /*存储日期数组区*/

void out_sbuf(void)      /*串口和中断初始化*/
{
    SCON=0x50;
    TMOD |=0x21;        /*4.8 kb/s 波特率*/
    PCON=0x00;
    TL1=0xf8;
    TH1=0xf8;
    TR1=1;
    /*ES=1; 串口允许中断*/
    /*EA=1; */
}

void delay_s(unsigned char k) /*短延时函数, k 是时常数*/
{
    unsigned char i;
    for (i=0; i<=k; ++i);
}

/*-----由串口发单个字符-----*/
void sendchar(uchar ch)
{
    SBUF=ch;
    while(TI==0);
    TI = 0;
    delay_s(10);        /*延时*/
}

/*有条件地读一个字符*/
uchar gethex_H(void)    /*超时退出*/
```

```

{
    unsigned int i=0;
    uchar c=0;
    while((RI==0)||(i<JS_MAX)) i++; /*超时增1*/
    c = SBUF;
    RI = 0;
    return(c);
}

/*-----接收$GPRMC 一帧数据---返回 0 正确, 返回 1 出错-----*/
uchar get_data(void) /*接收的数据在 x_data[]中*/
{
    uchar i,c_char;
    c_char=gethex_H(); /*接收帧头$*/
    x_data[0]=c_char;
    if (c_char != '$') goto end_getdata;
    c_char=gethex_H(); /*接收帧头 G*/
    x_data[1]=c_char;
    if (c_char != 'G') goto end_getdata;
    c_char=gethex_H(); /*接收帧头 P*/
    x_data[2]=c_char;
    if (c_char != 'P') goto end_getdata;
    c_char=gethex_H(); /*接收帧头 R*/
    x_data[3]=c_char;
    if (c_char != 'R') goto end_getdata;
    c_char=gethex_H(); /*接收帧头 M*/
    x_data[4]=c_char;
    if (c_char != 'M') goto end_getdata;
    c_char=gethex_H(); /*接收帧头 C*/
    x_data[5]=c_char;
    if (c_char != 'C') goto end_getdata;
    i=5;
    while( (c_char != 0x0D)||(c_char != 0x0A) )
    {
        c_char=gethex_H(); /*接收数据, 如果不等于“回车换行”, 则继续读*/
        i++;
        x_data[i]=c_char;
    }
    return 0;
end_getdata:
    return 1;
}

```

```

    }
    /*分离数据时, 如果数据完整, 则返回 0, 否则返回 1*/
    /*在数据分离完整的情况下, 数据存储在全局变量中*/
    uchar Read_data(void)          /*从 x_data[]中分离数据*/
    {
        uchar i,j,k;
        i=0;
        while ( x_data[i] != ',')    /*寻找第 1 个逗号*/
            {i++; }
        k=i+1;                        /*指向下一个数据*/
        for(j=0; j<6; j++)
            time[j]=x_data[k+j];     /*hhmmss 分离出的时间*/
        i=k+j+1;                     /*指向下一个数据*/
        while( x_data[i] != ',')     /*寻找第 2 个逗号*/
            {i++; }
        k=i+1;
        a_v=x_data[k];               /*存储定位标志 A 或 V */
        if(a_v=='V') goto cl_date;   /*如果定位无效, 则只有日期数据*/
        i=k+1;
        while( x_data[i] != ',')     /*寻找第 3 个逗号*/
            {i++; }
        k=i+1;
        for(j=0; j<9; j++)
            wd_data[j]=x_data[k+j];  /*存储纬度 ddmm.mmmm*/
        i=k+j+1;
        while( x_data[i] != ',')     /*寻找第 4 个逗号*/
            {i++; }
        k=i+1;
        wd_fx=x_data[k];             /*存储纬度的方向*/
        i=k+1;
        while( x_data[i] != ',')     /*寻找第 5 个逗号*/
            {i++; }
        k=i+1;
        for(j=0; j<10; j++)
            wd_data[j]=x_data[k+j];  /*存储经度 dddmm.mmmm*/
        i=k+j+1;
        while( x_data[i] != ',')     /*寻找第 6 个逗号*/
            {i++; }
        k=i+1;
        jd_fx=x_data[k];             /*存储经度的方向*/
        i=k+1;
    }

```

```

while( x_data[i] != ',')           /*寻找第7个逗号*/
{
    i++;
}
k=i+1;
while( x_data[i] != ',')           /*寻找第8个逗号*/
{
    i++;
}
k=i+1;
while( x_data[i] != ',')           /*寻找第9个逗号*/
{
    i++;
}
k=i+1;
for(j=0; j<6; j++)
    date[j]=x_data[k+j];           /*存储 ddmmyy(日月年)*/
return 0;

cl_date:
i=k+1; j=0;
while(j != 7)
{
    if(x_data[i]== ','){j++; i++; }
    else i++;
}
k=i+1;
for(j=0; j<6; j++)
    date[j]=x_data[k+j];           /*存储 ddmmyy(日月年)*/
return 1;                           /*表示只有日期和时间*/
}

void main(void)                     /*主函数部分通过循环接收 GPS 数据*/
{
    uchar numb;
    out_sbuf();                     /*串口初始化*/
    EA=0;                           /*关中断*/
    while(1)
    {
        while(get_data() == 0)       /*检索 GPS 数据(RMC)*/
        {
            numb=Read_data();         /*分离数据*/
            /*if(numb==0) send_numb(); 通过 GPRS 发送数据*/
        }
    }
}

```

第6章 数字电位器/电容器的使用与编程

6.1 X93154 低噪声低电压 32 抽头式数字电位器

6.1.1 硬件与功能描述

X93154 为 32 抽头式数控电位器(XDCP)。该器件内包含电阻阵列、滑动开关、控制单元和存储电阻位置的非易失性存储器等电路。滑动端位置由 3 线数字接口控制。

X93154 可广泛应用于偏压和增益控制、液晶显示器背光调节等领域。

1. 主要性能特点

- (1) 固态电位器, 总电阻值为 $50\text{ k}\Omega$;
- (2) 32 个滑动抽头点, 滑动点的位置存储于非易失性存储器中;
- (3) 31 个电阻单元, 具有温度补偿;
- (4) 点对点电阻范围为 $\pm 30\%$, 终端电压为 $0\sim V_{cc}$;
- (5) 低电压 CMOS 模式, V_{cc} 为 $3\text{ V}\pm 10\%$;
- (6) 有效电流最大为 $250\text{ }\mu\text{A}$, 待机电流最大为 $1\text{ }\mu\text{A}$;
- (7) 每位可允许 20 000 次数据擦写;
- (8) 3 线串行接口;
- (9) 数据保存至少 10 年;
- (10) 器件工作温度为 $-40\sim +85^\circ\text{C}$ 。

2. 内部结构与引脚说明

X93154 电位器由一个包含 31 个电阻单元的电阻阵列和一个滑动端开关网络组成。滑动端的位置由 $\overline{\text{CS}}$ 、 $\text{U}/\overline{\text{D}}$ 和 INC 输入端控制。其结构如图 6-1(a)所示。

X93154 的输入控制计数器、数字译码器、非易失性存储器和电阻阵列控制部分就像一个升/降计数器。这个计数器的输出被译码而接通一个单接点的电子开关, 以便把电阻阵列上的滑动点连接到输出端。在适当的条件下, 计数器的内容可存储在非易失性存储器中以便后期使用。电阻阵列的两个端点以及每个电阻之间都有一个电子开关, 以便把该点的电位传输至滑动输出端。

当滑动端位于任一固定端点时, 就像等效的机械滑动端一样, 不会移动到超出终端位置。当计数器达到一个极端时, 不会循环。当器件被断电时, 最后存储的计数器状态将被维持在非易失性存储器中。电源恢复时存储器中的内容被调出, 这样就可保证电阻位置与上次一样。

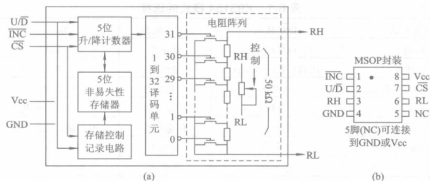


图 6-1 X93154 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

X93154 数字电位器的引脚排列如图 6-1(b)所示。其中:

(1) INC 是增加输入脚。增加输入脚是负边沿触发, $\overline{\text{INC}}$ 将使滑动端向计数器增加或减少的方向移动, 移动的方向由 $\text{U}/\overline{\text{D}}$ 端输入的逻辑电平决定。

(2) $\text{U}/\overline{\text{D}}$ 是升/降方向控制脚。当该脚为“1”时表示上升, 当该脚为“0”时表示下降。

(3) RH 是 X93154 的电阻高端, 等效于机械电位器的固定端。其最小电压是 0 V, 而最大电压是 V_{cc} 。

(4) RL 是 X93154 的电阻低端, 等效于机械电位器的固定端。其最小电压是 0 V, 而最大的电压是 V_{cc} 。注意, RL 和 RH 只是规定了由 $\text{U}/\overline{\text{D}}$ 输入端选择的关于滑动端方向的相对位置。

(5) $\overline{\text{CS}}$ 是片选输入端。当 $\overline{\text{CS}}$ 输入为低时, 器件被选中; 当 $\overline{\text{CS}}$ 变为高且 $\overline{\text{INC}}$ 输入端也为高时, 在储存操作完成后 X93154 将处于低功耗的等待方式, 直到器件再次被选中。

(6) V_{cc} 是电源电压端。

3. X93154 的操作

$\overline{\text{INC}}$ 、 $\text{U}/\overline{\text{D}}$ 和 $\overline{\text{CS}}$ 三个输入端控制滑动端沿着电阻阵列移动。只有 $\overline{\text{CS}}$ 置低(器件被选中), 才能使 $\overline{\text{INC}}$ 和 $\text{U}/\overline{\text{D}}$ 输入端接收信号。在 INC 输入端由高至低的变化将增加或减少(这取决于 $\text{U}/\overline{\text{D}}$ 的输入状态)一个 5 位计数器的值。这个计数器的输出经 32 选 1 译码去控制 32 个抽头(就像机械电位器的滑动头一样)。

只要当 $\overline{\text{CS}}$ 转变为高, $\overline{\text{INC}}$ 输入端也为高时, 计数器的值就被储存在非易失性存储器中。为避免上电时的意外存入, 在初次上电期间 $\overline{\text{CS}}$ 脚与 V_{cc} 必须为高。在进行多次写操作时, V_{cc} 不得降低至低于其初始值, 在开路时, $\overline{\text{CS}}$ 引脚被一个内部 30 k Ω 电阻上拉到 V_{cc} , 以便保证器件的可靠性。

在上电时 $\overline{\text{CS}}$ 引脚必须保持为高, 这将存储(或预置)一个值在非易失性存储器中, 可以解决由于温度漂移引起的系统参数变化。当 $\overline{\text{CS}}$ 保持为低时, $\text{U}/\overline{\text{D}}$ 的状态可以改变, 这就允许主系统可以滑动上升或下降以达到适合的微调为止。

4. 模式的选择

X93154 模式的选择如表 6-1 所示。

表 6-1 X93154 模式的选择

$\overline{\text{CS}}$ 状态	$\overline{\text{INC}}$ 状态	U/D 状态	模 式
低电平	由高向低(下降沿)	高电平	滑动向上
低电平	由高向低(下降沿)	低电平	滑动向下
由低向高(上升沿)	高电平	任意状态	存储滑动位置
高电平	任意状态	任意状态	进入待机状态
由低向高(上升沿)	低电平	任意状态	不存储, 返回待机状态
由高向低(下降沿)	低电平	高电平	滑动向上(无补偿)
由高向低(下降沿)	低电平	低电平	滑动向下(无补偿)

6.1.2 应用电路与编程

1. 典型应用电路

数字电位器(XDCP)具备如下三个应用优势:

- (1) 固态电位器的可变性和可靠性;
- (2) 基于计算机数控的灵活性;
- (3) 用于存储多个电位器设置的记忆性。

图 6-2 是两种放大器增益控制电路, 其中图(a)是常用的三运放组合数据放大器, 图(b)是常见的单电压可变增益放大器。

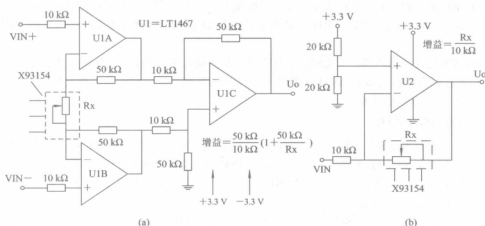


图 6-2 两种数控放大器电路

(a) 数据放大器; (b) 反相放大器

2. 编程方法

对于 X93154 数字电位器, 只要定义单片机的 3 个 I/O 口与数字电位器的相接关系就可进行编程。如果单片机选择 STC89C52, 将 P2.0、P2.1 和 P2.2 分别与 X93154 的第 1 脚($\overline{\text{INC}}$)、第 2 脚(U/D)与第 7 脚($\overline{\text{CS}}$)相接, 则按照表 6-1 的工作模式, 由 C51 所实现的源程序如下:

```
#include <reg52.h>
```

```

sbit inc=P2^0;           /*定义上升/降低引脚*/
sbit u_d=P2^1;           /*定义上升/降低控制方向引脚*/
sbit cs =P2^2;           /*定义器件片选引脚*/
sbit up_key =P2^3;       /*定义上升的按键，为“0”有效*/
sbit down_key=P2^4;      /*定义下降的按键，为“0”有效*/

#define uchar unsigned char
#define MAX_N 0x32       /*定义最大的计数值*/
uchar count=0;           /*定义的计数变量*/
void delay(unsigned int k) /*延时函数，k是时常数*/
{
    unsigned int i;
    for (i=0; i<=k; ++i);
}

/*up_down=0是下滑动，up_down=1是上滑动*/
void slow_u_d(uchar up_down)
{
    cs=1; inc=1; u_d=1;   /*进入待机状态*/
    if(up_down==1)        /*向上滑动*/
    {
        if(count < MAX_N)
        {
            count++;      /*计数器增1*/
            cs=0;          /*选中器件*/
            u_d=1;         /*将方向置成向上*/
            inc=0;         /*i+生下降沿，往上滑动一步*/
            delay(20);     /*延时*/
            inc=1;         /*为存储做准备*/
            cs=1;          /*i+生上升沿，将滑动位置存储在非易失性存储器中*/
        }
        else count=MAX_N;
    }
    else if(up_down==0)   /*向下滑动*/
    {
        if((count > 0)&&(count <= MAX_N))
        {
            count--;      /*计数器减1*/
            cs=0;          /*选中器件*/
            u_d=0;         /*将方向置成向下*/
            inc=0;         /*i+生下降沿，往下滑动一步*/
            delay(20);     /*延时*/
            inc=1;         /*为存储做准备*/
        }
    }
}

```



```

        cs=1;          /*产生上升沿, 将滑动位置存储在非易失性存储器中*/
    }
    else count=0;
}
}
void main(void)        /*主函数部分演示升和降的过程*/
{
    while (1)
    {
        if(up_key==0)   /*如果有按键, 则“动作”*/
        {
            slow_u_d(0x1); /*向上滑动*/
            delay(0x200);  /*延时去抖*/
        }
        if(down_key==0) /*如果有按键, 则“动作”*/
        {
            slow_u_d(0x0); /*向下滑动*/
            delay(0x200);  /*延时去抖*/
        }
    }
}
}

```

6.2 X9221A 双数控 64 抽头数字电位器

6.2.1 硬件与功能描述

X9221A 是把两个数字电位器(XDCP)集成在一个单片 CMOS 电路中的数控器件。它由 63 个串联在一个阵列的电阻单元组成, 每个单元之间都通过开关连接到各自的抽头滑动点上。阵列中的电阻位置可由用户通过 2 线总线接口控制。每个电位器均配有一个易失性的滑动端计数寄存器 and 四个非易失性的辅助数据寄存器。辅助数据寄存器可由用户直接写入或读出。计数寄存器(WCR)上电时, 自动将辅助数据寄存器 0 的内容重新调入 WCR 中, 以恢复断点前的电阻位置。

XDCP 可用作一个三端的电位器或一个二端的可变电阻使用, 应用范围广泛。

1. 主要性能特点

- (1) 两个 Xicor 数控电位器集成在一个封装内;
- (2) 2 线串行接口(类似于 I²C 接口);
- (3) 8 个数据寄存器可直接写入滑动端位置, 也可读出滑动端位置;
- (4) 电位器的操作通过指令完成;
- (5) 电阻阵列值有: 2 k Ω 、10 k Ω 和 50 k Ω ;
- (6) 分辨率: 每个电位器 64 个抽头(6 位二进制状态);
- (7) 每位可允许 100 000 次以上数据擦写, 数据可保存至少 10 年;

(8) 电源电压: $5\text{V} \pm 10\%$;

(9) 器件工作温度: $0 \sim 70^\circ\text{C}$ (民品), $-40 \sim +85^\circ\text{C}$ (工业级)。

2. 内部结构与引脚说明

X9221A 的内部结构如图 6-3(a)所示。该器件包含两个电阻阵列(POT0、POT1)。每个阵列包含 63 个串联连接的分立的电阻段。每个阵列的物理端等效于一个机械电位器的固定端(VH/RH 和 VL/RL)。在每个阵列的两个终端以及每个电阻段之间是一个连接到滑动输出端(VW/RW)的 FET 开关。在每个单独的阵列中同一时间只有一个开关可以接通, 这些开关由滑动端计数寄存器 WCR 控制, 即 WCR 中的低 6 位被译码并完成 64 选 1 的开关控制。

X9221A 的 XDCP 等效电位器如图 6-3(b)所示。

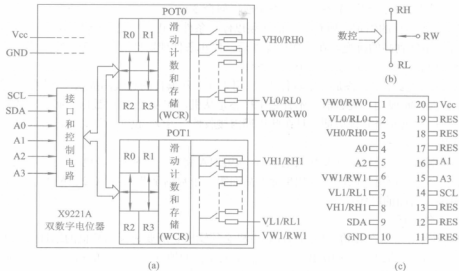


图 6-3 内部结构、等效电位器与引脚排列

(a) 内部组成; (b) 等效电位器; (c) 引脚排列

X9221A 数字电位器的引脚排列见图 6-3(c)。其中:

- (1) SCL 是串行接口的时钟输入端。SCL 用于同步数据的时钟。
- (2) SDA 是串行数据端。SDA 是一个双向引脚, 用于向器件输入或输出数据。它是一个漏极开路输出, 可以与多路漏极开路输出或集电极开路输出相接, 但要接合适的上拉电阻。
- (3) A0~A3 是器件从地址输入端。从地址的串行数据流必须与输入地址相匹配才能开始与该 X9221A 的通信。
- (4) VL0/RL0、VL1/RL1、VH0/RH0 和 VH1/RH1 是数字电位器的两个固定端引脚(相当于机械电位器的两个固定端), 如图 6-3(b)中的 RH 和 RL。
- (5) VW0/RW0 和 VW1/RW1 是数字电位器的两个滑动端(相当于机械电位器的滑动端), 如图 6-3(b)中的 RW。
- (6) Vcc 是电源端, 通常接 5V 。
- (7) RES 是保留端, 悬空不用。

3. 器件的操作

1) 串行接口

X9221A 支持双向总线的协议(类似于 I²C 协议)。该协议定义任何向总线发送数据的器件为发送器,接收数据的器件为接收器,传送控制信号的器件为主机,而被控制的器件为从机。主机总是启动数据传输过程,X9221A 在所有应用中只能作为从机。

(1) 时钟和数据的约定。只有在时钟(SCL)为低电平时,数据线(SDA)上的数据状态才允许改变。当 SCL 为高时,SDA 的状态改变只能引起器件“开始”或“停止”操作。

(2) 开始条件的约定。向 X9221A 发出的所有命令都是由开始条件引起的。它是一个当 SCL 为高时,在 SDA 线上由高到低的跳变(X9221A 不断监视 SDA 和 SCL 线上的“开始条件”,并且在没有遇到这个条件之前不响应任何命令)。

(3) 停止条件的约定。所有的通信必须以停止条件终止。它是一个当 SCL 为高时,在 SDA 线上由低到高的跳变。在一次读操作后,停止条件使器件进入电源等待方式。只有在发送器释放总线后,才能发送一个停止条件。

(4) 应答的约定。应答是一个软件协议,用来在主从器件的总线间提供一个“握手信号”,以表示数据接收成功。发送器件不管是主机还是从机,在发送 8 位数据后必须释放总线,并使主机产生第 9 个时钟周期,且在此周期内接收器将 SDA 拉低,作为它已接收到 8 位数据的应答。

在识别出一个开始条件及其从地址后,X9221A 将以一个应答作为响应,并在成功地接收到命令字节后它将再次应答。如果命令后面跟一个数据字节,则 X9221A 将响应一个最终应答(参见后面的时序)。

2) 工作流程

(1) 器件寻址。在开始条件的后面,主机必须输出它所访问的从机的地址,从机的高 4 位地址是器件的识别码(0101B),低 4 位是从器件的地址码。器件的地址码(物理地址)由 A0~A3 输入端的状态来确定,如图 6-4(a)所示。



图 6-4 X9221A 数据传输格式

(a) 地址构成; (b) 指令结构

(2) 指令结构。X9221A 的指令是一个字节。高 4 位是指令,低 4 位是选择辅助寄存器的信息。其格式如图 6-4(b)所示。I3~I0 是指令码(高 4 位,见表 6-2);第 6 位(P0)用来选择两个电位器的哪一个将受指令控制(P0=0 是 POT0, P0=1 是 POT1);最后 2 位(R1 和 R0)用来选择 4 个寄存器中的一个(即 00~11 选择 R0~R3)。当一条与寄存器有关的指令发出时,该寄存器将被控制。

(3) 操作指令与时序。X9221A 的操作指令如表 6-2 所示。在 9 条指令中,有 4 条是二字节时序(器件寻址和指令),如图 6-5 所示。这 4 条指令 I3~I0 是: 1101、1110、0001 和 1000。从一个数据寄存器到一个 WCR 间的传输实质上相当于对一个静态 RAM 的一次写入。

从 WCR 的当前滑动端位置到一个数据寄存器间的传输是一次对非易失性存储器的写入。这种操作可以为发生在任何时间对器件的操作(滑动端对这种作用将有一点延迟)。

表 6-2 X9221A 的操作指令

指 令	指令格式								操作含义
	I3	I2	I1	I0	0	P0	R1	R0	
读 WCR	1	0	0	1	0	I/O	N/A	N/A	读出由 P0 指定的计数器内容
写 WCR	1	0	1	0	0	I/O	N/A	N/A	写入由 P0 指定的计数器值
读数据寄存器	1	0	1	1	0	I/O	I/O	I/O	读出 POR1R0 指的寄存器内容
写数据寄存器	1	1	0	0	0	I/O	I/O	I/O	写入 POR1R0 指的寄存器内容
XFR 内容到 WCR	1	1	0	1	0	I/O	I/O	I/O	传 POR1R0 指的 XFR 内容到 WCR
XFR WCR 到寄存器	1	1	1	0	0	I/O	I/O	I/O	传 P0 指的 WCR 到 R1R0 的寄存器
全局 XFR 到 WCR	0	0	0	1	N/A	N/A	I/O	I/O	传 POR1R0 指的 XFR 内容到 WCR
全局 WCR 到 XFR	1	0	0	0	N/A	N/A	I/O	I/O	传 POR1R0 指的 WCR 内容到 XFR
增加/减少	0	0	1	0	0	I/O	N/A	N/A	增加/减少由 P0 指定的 WCR

注: N/A 是不关心位。

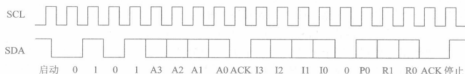


图 6-5 二字节时序

在 X9221A 的指令集中，有 4 条指令是一个三字节的操作时序。这些指令如下：

- (1) 读 WCR，即读出选定电位器的当前滑动端的位置；
- (2) 写 WCR，即改变选定电位器的当前滑动端的位置；
- (3) 读数据寄存器，即读出选定的非易失性寄存器的内容；
- (4) 写数据寄存器，即写一个新的值到选定的数据寄存器中。

其操作时序如图 6-6 所示。

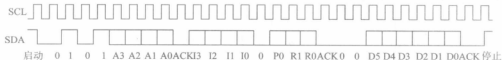


图 6-6 三字节时序

增加/减少指令与其他指令不同。一旦这个指令发出而 X9221A 已经用一个应答来响应，主机就能够以时钟来触发选定的滑动端升或降一个电阻段。这样就为主机提供了一个精细的调整能力。当 SDA 为高时，每一个 SCL 时钟脉冲将使选定的滑动端向 VH/RH 端移动一个电阻段。相似地当 SDA 为低时，每个 SCL 时钟脉冲将使选定的滑动端向 VL/RL 端移动一个电阻段，见图 6-7 中的“↑”升和“↓”降。

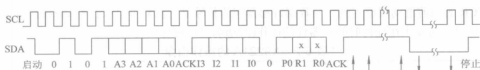


图 6-7 增加/减少时序

3) 相关寄存器说明

在 X9221A 的两个 XDCP 电位器中, 每个 XDCP 电位器有一个滑动端计数寄存器 and 4 个辅助数据寄存器。这些寄存器的内容直接影响 XDCP 的工作。

(1) 滑动端计数寄存器(WCR)。X9221A 包括两个滑动端计数寄存器(WCR), 每个 XDCP 电位器各有一个。WCR 可以被认为是一个 6 位并行和串行装载且具有输出译码的计数器。WCR 的内容可用如下四种方法来改变:

- ① 可以由主机通过“写 WCR”指令来直接写入(串行装载);
- ② 可以通过“XFR 数据寄存器”指令把四个辅助数据寄存器之一的内容直接写入(并行装载);
- ③ 可以通过增加/减少指令一步一步地修改;
- ④ 可以在上电时装入它的数据寄存器 0(R0)的内容。

WCR 是一个易失性寄存器, 虽然当 X9221A 断电时它的内容将丢失, 但在上电时却能自动地把 R0 寄存器的内容装入(必须注意这个值可能与断电时的值不同)。

(2) 辅助数据寄存器。每个电位器有 4 个(R0~R3)非易失性数据寄存器, 这些寄存器可以被主机直接读出或写入, 而且数据可以在 4 个数据寄存器和 WCR 之间传输。必须注意, 在这些寄存器中的任何一个改变数据的操作都是非易失性的操作, 将花去约 10 ms 的时间。

如果在应用中不需要对电位器的位置进行存储, 则这些寄存器可被用作通用的存储单元, 可以储存系统参数或用户数据。

6.2.2 应用电路与编程

由 89C52 单片机、X9221A 和运算放大器构成的数控反相比例放大器如图 6-8 所示。其中, Rx 是 X9221A 的一个数字电位器, 用来对放大器进行调零。Ry 是 X9221A 的另一个数字电位器, 用来实现增益控制。

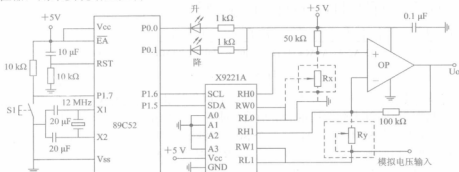


图 6-8 X9221A 的一种应用电路

根据 X9221A 的工作原理和图 6-8 的应用特点, 用 C51 实现的相关函数如下:

```

#include <reg51.h>

sbit K1 = P1^7;          /*按键引脚*/
sbit SCL= P1^6;          /*串行时钟引脚*/
sbit SDA= P1^5;          /*串行数据引脚*/
sbit led_s=P0^0;         /*升指示灯引脚*/
sbit led_d=P0^1;         /*降指示灯引脚*/

void delay(unsigned int x) /*延时函数*/
{
    unsigned int i;
    for (i=0; i<x; i++);
}

void start(void)          /*启动函数*/
{
    SCL=1; SDA=1;         /*时钟线=1, 数据线=1*/
    delay(5); SDA=0;
    delay(5); SCL=0;
}

void stop(void)           /*停止函数*/
{
    SDA=0; SCL=0;         /*数据线=0, 时钟线=0*/
    delay(5); SCL=1;
    delay(5); SDA=1;
}

void send_8(unsigned char data8) /*写 8 位数据函数*/
{
    unsigned char dat,i;
    dat=data8;            /*发送数据*/
    for(i=0; i<8; i++)
    {
        if(dat&0x80) SDA=1; /*高位在前*/
        else SDA=0;        /*先准备好数据*/
        delay(2); SCL=1;   /*时钟线=1*/
        delay(2); SCL=0;   /*时钟线=0*/
        delay(2); dat<<=1; /*左移 1 位*/
    }
}

unsigned char read_data(void) /*读 8 位数据函数*/
{
    unsigned char i,adat=0;

```

```

for(i=0; i<8; i++)
{
    adat<=1;
    SCL=1; delay(2);
    if(SDA==1)  adat=adat | 0x01;
    else  adat=adat | 0x00;
    SCL=0; delay(2);
}
return adat;
}

void read_ack(void)          /*读 ACK 函数*/
{
    unsigned char ack;
    SDA=1; delay(2); SCL=1;
    if(SDA==0)  ack=0;
    else  ack=1;
    SCL=0;
}

void write_ack(void)        /*发送 ACK, 通知从机收到数据*/
{
    SCL=0; delay(2);
    SCL=1; delay(2);
    SDA=0; delay(2);      /*回答*/
    SCL=0;
}

/*-----
功能为向 X9221A 写入两个字节, 参数 address、cd 是写入的数据
-----*/

void Write_X9221A_8(address,cd)
unsigned char address,cd;
{
    start();              /*启动*/
    send_8(address);      /*发送器件的从地址*/
    read_ack();           /*读 ACK 函数*/
    send_8(cd);           /*发送命令*/
    read_ack();           /*读 ACK 函数*/
    stop();               /*停止条件*/
}

/*-----
操作: 向 X9221A 操作三个字节, 写两个字节, 读一个字节作为返回
-----*/

```

```

unsigned char read_1_write_2(address,cd)
unsigned char address,cd;
{
    unsigned char dat;
    start();                /*启动*/
    send_8(address);        /*发送器件的从地址*/
    read_ack();             /*读 ACK 函数*/
    send_8(cd);             /*发送命令*/
    read_ack();             /*读 ACK 函数*/
    dat=read_data();        /*读一个字节数据*/
    write_ack();            /*通知从机数据收到*/
    stop();                 /*停止条件*/
    return dat;
}
/*-----

```

操作：向 X9221A 写三个字节，即 address、cd 和 byte

```

-----*/
void write_3(address,cd,byte)
unsigned char address,cd,byte;
{
    start();                /*启动*/
    send_8(address);        /*发送器件的从地址*/
    read_ack();             /*读 ACK 函数*/
    send_8(cd);             /*发送命令*/
    read_ack();             /*读 ACK 函数*/
    send_8(byte);           /*发送数字*/
    read_ack();             /*读 ACK 函数*/
    stop();                 /*停止条件*/
}
/*-----

```

操作：上升或下降一个电阻挡，写两个字节，u_d = 0 时下降，u_d = 1 时上升

```

-----*/
void up_down(address,cd,u_d)
unsigned char address,cd,u_d;
{
    start();                /*启动*/
    send_8(address);        /*发送器件的从地址*/
    read_ack();             /*读 ACK 函数*/
    send_8(cd);             /*发送命令*/

```



```

SDA=1; /*释放数据线*/
read_ack(); /*读 ACK 函数*/
if(u_d==0)
{ SDA=0;
  SCL=1; delay(10);
  SCL=0; delay(10);
  led_s=1; led_d=0; /*将向下的灯点亮*/
}
else if(u_d==1)
{ SDA=1;
  SCL=1; delay(10);
  SCL=0; delay(10);
  led_d=1; led_s=0; /*将向上的灯点亮*/
}
SDA=0;
stop(); /*停止条件*/
}
void main(void) /*示范主函数*/
{
  unsigned char wcr,R0;
  led_s=1; led_d=1;
  R0=read_1_write_2(0x50,0xb0); /*读 Rx 的 R0, 判断是否超界*/
  if(R0>64) write_3(0x50,0xc0,32); /*给 Rx 的 R0 写初值为 32*/
  R0=read_1_write_2(0x50,0xb4); /*读 Ry 的 R0, 判断是否超界*/
  if(R0>64) write_3(0x50,0xc4,32); /*给 Ry 的 R0 写初值为 32*/
  Write_X9221A_8(0x50,0xd0); /*写 R0 到 Rx 的 WRC*/
  Write_X9221A_8(0x50,0xd4); /*写 R0 到 Ry 的 WRC*/
  While(1)
  {
    if(S1==0) /*S1=0,动作一次*/
    {
      up_down(0x50,0x20,0x00); /*使 Rx 阻值减少一挡*/
      Write_X9221A_8(0x50,0xc0); /*将 WRC 内容写入 R0*/
      up_down(0x50,0x24,0x01); /*使 Ry 阻值增加一挡*/
      Write_X9221A_8(0x50,0xc4); /*将 WRC 内容写入 R0*/
    }
    wcr=read_1_write_2(0x50,0x90); /*读 Rx 的 WRC 值*/
  }
}

```

6.3 X9318 数控 100 抽头数字电位器

6.3.1 硬件与功能描述

X9318 为 100 个抽头的数字电位器(XDCP)。X9318 内部包含电阻阵列、滑动开关、控制单元和非易失性存储器,滑动端的位置由 3 线接口电路控制。

该器件可广泛用于液晶显示器偏压控制、直流电偏压调整、激光二极管偏压控制和电压稳压器输出控制等领域。

1. 主要性能特点

- (1) 固态电位器的总电阻值为 $10\text{ k}\Omega$;
- (2) 100 个滑动抽头点,滑动点的位置存储于非易失性存储器中;
- (3) 99 个电阻单元,具有温度补偿;
- (4) 点对点电阻范围为 $\pm 20\%$,端点电压为 $0\sim +8\text{ V}$;
- (5) 低电压 CMOS 结构, V_{CC} 为 $5\text{ V}\pm 10\%$;
- (6) 工作电流最大为 3 mA ,待机电流最大为 1 mA ;
- (7) 每位可允许 100 000 次数据擦写;
- (8) 3 线串行接口;
- (9) 数据保存至少 100 年;
- (10) 器件工作温度为 $0\sim +70^\circ\text{C}$ (民用), $-40\sim +85^\circ\text{C}$ (工业)。

2. 内部结构与引脚说明

X9318 数字电位器由一个包含 99 个电阻单元的电阻阵列和一个滑动端开关网络组成。滑动端的位置由 $\overline{\text{CS}}$ 、 $\text{U}/\overline{\text{D}}$ 和 INC 输入端控制。其内部结构如图 6-9(a)所示。

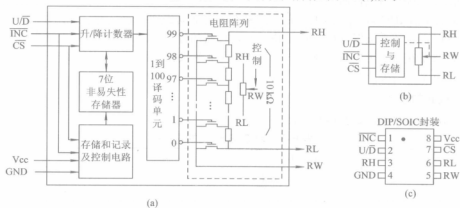


图 6-9 X9318 的内部结构、等效电路与引脚排列

(a) 内部结构; (b) 等效电路; (c) 引脚排列

控制部分的工作就像一个升/降计数器。这个计数器的输出被译码而接通一个单接点的电子开关,以便把电阻阵列上的一个点连接到滑动输出端。计数器的内容可以存储在非易失性存储器中。

滑动端就像等效的机械滑动端一样, 见图 6-9(b)。它不会移动到超出 RH 或 RL 末端位置, 即当计数器达到一个极端时不会循环。

当滑动端改变抽头位置时, 器件的电子开关以“先接通后断开”的方式工作。当器件被断电后, 存储的滑动端位置将被维持在非易失性存储器中, 电源恢复时存储器中的内容被重新调用, 使滑动点的位置保持与断电前一致。

X9318 数字电位器的引脚排列如图 6-9(c)所示。其中:

(1) $\overline{\text{INC}}$ 是增加输入脚。增加输入脚是下降沿触发, $\overline{\text{INC}}$ 将使滑动端向计数器增加或减少的方向移动, 移动的方向由 $\text{U}/\overline{\text{D}}$ 端输入的逻辑电平决定。

(2) $\text{U}/\overline{\text{D}}$ 是升/降方向控制脚。当该脚为高电平时上升, 当该脚为低电平时下降。

(3) RH 是 X9318 的电阻高端, 等效于机械电位器的一个固定端。

(4) RL 是 X9318 的电阻低端, 等效于机械电位器的一个固定端。

(5) RW 是 X9318 的电阻滑动端, 等效于一个机械电位器的可移动端。滑动端在电阻阵列中的位置由控制输入脚决定, 滑动端电阻值通常为 $40\ \Omega$ 。

(6) $\overline{\text{CS}}$ 是片选输入端, 低电平有效。当 $\overline{\text{CS}}$ 变为高电平, 且 $\overline{\text{INC}}$ 输入端也为高电平时, 当前计数器的值就被存储在非易失性存储器中。当存储操作完成后, X9318 将立即处于低功耗的等待方式, 直到器件再次被选中。

(7) Vcc 是电源电压端。

(8) GND 是参考地。

3. X9318 的操作过程

$\overline{\text{INC}}$ 、 $\text{U}/\overline{\text{D}}$ 和 $\overline{\text{CS}}$ 三个输入信号控制滑动端沿着电阻阵列移动。只有 $\overline{\text{CS}}$ 置低(器件被选中), 才能使 $\overline{\text{INC}}$ 和 $\text{U}/\overline{\text{D}}$ 输入端接收信号。在 $\overline{\text{INC}}$ 输入端由高至低的变化将增加或减少(这取决于 $\text{U}/\overline{\text{D}}$ 的输入状态)一个 7 位计数器的值。这个计数器的输出经 100 选 1 译码去控制 100 个抽头(就像机械电位器的滑动头一样)。

当 $\overline{\text{CS}}=1$ 和 $\overline{\text{INC}}=1$ 时, 计数器的值就被存储在非易失性存储器中。当滑动端移动一个新的位置时, 系统必须保持 $\overline{\text{CS}}=1$ 和 $\overline{\text{INC}}=1$, 使新的位置存入非易失性存储器中。这种操作将允许系统在每次上电时, 预置一个值存储在非易失性存储器中。当系统重新上电时, 可保证电位器不会失控。

4. 模式的选择

X9318 模式的选择如表 6-3 所示。

表 6-3 X9318 模式的选择

$\overline{\text{CS}}$ 状态	$\overline{\text{INC}}$ 状态	$\text{U}/\overline{\text{D}}$ 状态	模 式
低电平	由高向低(下降沿)	高电平	滑动向上
低电平	由高向低(下降沿)	低电平	滑动向下
由低向高(上升沿)	高电平	低电平	存储滑动位置到存储器中
高电平	高电平	高电平	进入待机状态
由低向高(上升沿)	低电平	低电平	不存储, 返回待机状态
由高向低(下降沿)	低电平	高电平	滑动向上(无补偿)
由高向低(下降沿)	低电平	低电平	滑动向下(无补偿)

6.3.2 应用电路与编程

1. 典型电路

X9318 数字电位器的几种典型用法如图 6-10 所示。

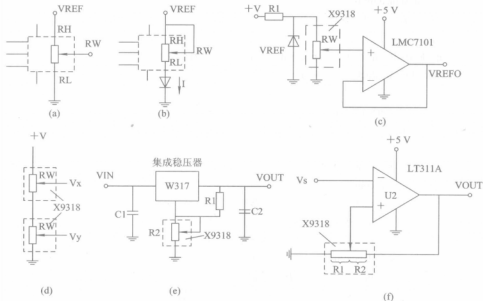


图 6-10 X9318 数字电位器的几种典型用法

(a) 基本分压; (b) 可变电流; (c) 缓冲基准电源; (d) 级联接法; (e) 可调稳压电源; (f) 滞后比较器

图 6-10 中, 图(a)是采用电位器三端口分压的基本电路; 图(b)是采用电位器二端口的接法; 图(c)是可调基准电压缓冲电路, 通过 X9318 的调整可实现数控基准电源; 图(d)是采用两个 X9318 相串联的结构; 图(e)是实现数控稳压电源的应用电路, 在输入 VIN 正常(输入电压比输出要大 2 V 压差)的情况下, $V_{OUT} \approx 1.25(1 + R2/R1)$; 图(f)是一种带滞后功能的比较器电路, 调节 X9318 的滑动点位置可改变 R1、R2 的分配比例, 从而达到有滞后的比较输出。

2. 编程方法

通过单片机的编程控制 X9318 极为简单。设单片机(STC89C52RC)的 P1.5、P1.6、P1.7 分别接 X9318 的 INC、U/D 和 CS。根据数字电位器的工作原理和表 6-3 的约定, 用 C51 所实现的相关源程序如下:

```
#include <reg52.h>

sbit inc=P1^5;          /*定义上升/降低引脚*/
sbit u_d=P1^6;          /*定义上升/降低控制方向引脚*/
sbit cs=P1^7;           /*定义器件片选引脚*/
sbit up_key=P2^0;        /*定义上升的按键, 为“0”时有效*/
sbit down_key=P2^1;      /*定义下降的按键, 为“0”时有效*/

#define uchar unsigned char
```

```

#define MAX_N 100          /*定义最大的计数值*/
uchar count=0;            /*定义的计数变量*/
void delay(unsigned int k) /*延时函数, k 是时常数*/
{
    unsigned int i;
    for(i=0; i<k; i++);
}
/*up_down=0, 是下滑动; up_down=1, 是上滑动, 为小步子滑动*/
void slow_u_d(uchar up_down)
{
    cs=1; inc=1; u_d=1;    /*进入待机状态*/
    if(up_down==1)         /*向上滑动*/
    {
        if(count < MAX_N)
        {
            count++;       /*计数器增 1*/
            cs=0;          /*选中器件*/
            u_d=1;         /*将方向置成向上*/
            inc=0;          /*产生下降沿, 往上滑动一步*/
            delay(20);     /*延时*/
            inc=1;         /*为存储做准备*/
            cs=1;          /*产生上升沿, 将滑动位置存储在非易失性存储器中*/
        }
        else count=MAX_N;
    }
    else if(up_down==0)
    {
        if((count > 0) & (count <= MAX_N))
        {
            count--;       /*计数器减 1*/
            cs=0;          /*选中器件*/
            u_d=0;         /*将方向置成向下*/
            inc=0;          /*产生下降沿, 往下滑动一步*/
            delay(20);     /*延时*/
            inc=1;         /*为存储做准备*/
            cs=1;          /*产生上升沿, 将滑动位置存储在非易失性存储器中*/
        }
        else count=0;
    }
}

/*up_down=0, 是下滑动; up_down=1, 是上滑动。step 是步长, 规定小于 10*/
void up_down(uchar up_down, uchar step) /*是大步子滑动*/

```

```

    {   uchar n,i;
        if(step>10) n=10;
        else if(step<1) n=1;
        if(up_down==0)           /*向下滑动*/
        {
            for (i=0; i<n; i++)   /*向下移动 n 次*/
            {
                slow_u_d(0);
                delay(0x10);
            }
        }
        else if (up_down==1)      /*向上滑动*/
        {
            for (i=0; i<n; i++):  /*向上移动 n 次*/
            {
                slow_u_d(1);
                delay(0x10);
            }
        }
    }
}

void main(void)                  /*主函数部分演示升和降的过程*/
{
    while (1)
    {
        if(up_key==0)            /*如果有按键，则“动作”*/
        {   up_down(1,0x04);      /*向上滑动 4 次*/
            delay_s(0x200);        /*延时去抖*/
        }
        if(down_key==0)          /*如果有按键，则“动作”*/
        {   up_down(0,0x04);      /*向下滑动 4 次*/
            delay_s(0x200);        /*延时去抖*/
        }
    }
}

```

6.4 MCP41xxx 系列低功耗 256 抽头数字电位器

6.4.1 硬件与功能描述

机械式电位器通常用来调整系统参考电压、增益误差和偏置电压误差。数字电位器可以用来完成相同的任务，而且还能提供额外的数字调整控制功能。MCP41xxx 和 MCP42xxx

数字电位器系列器件可以在很大程度上像机械式电位器那样使用,这是因为对于含单个电位器的器件(MCP41010、MCP41050 和 MCP41100),有三个电阻端子,而对于含两个电位器的器件(MCP42010、MCP42050 和 MCP42100),则有六个电阻端子,就像机械电位器一样。

MCP41xxx系列器件是具有256个抽头的数字电位器(XDCP)。该系列电阻有10 k Ω 、50 k Ω 和100 k Ω 几种,内部包含电阻阵列、滑动开关、控制单元和16位存储器。滑动端的位置由SPI总线控制。

每次上电或重新复位“数据字节”的数据被初始化为80H(即电位器的滑动端处在中心位置)。

MCP41xxx系列器件采用CMOS工艺,功耗极低,被广泛地应用于仪器仪表和精密电压或电流控制系统中。

1. 主要性能特点

- (1) 该系列总电阻有 10 k Ω 、50 k Ω 和 100 k Ω 几种;
- (2) 256个滑动抽头点,滑动点的位置存储于内部存储器中;
- (3) 255个电阻单元,具有温度补偿;
- (4) 采用 SPI 串行接口;
- (5) 宽电压 CMOS 技术,最大工作电流为 500 μ A,最大静态电流为 1 μ A;
- (6) 电源电压范围为 2.7~5.5 V;
- (7) 器件工作温度为-40~+85 $^{\circ}$ C(工业)、-40~+125 $^{\circ}$ C(军品)。

2. 内部结构与引脚排列

MCP41xxx 系列数字电位器由一个包含 255 个电阻单元的电阻阵列和一个滑动端开关网络组成。滑动端的位置由 $\overline{\text{CS}}$ 、SI 和 SCK 3 线输入信号控制。其结构如图 6-11(a)所示。

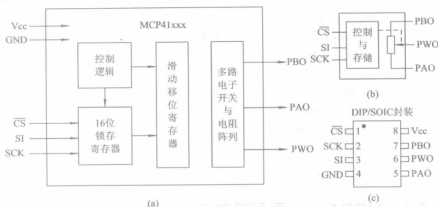


图 6-11 MCP41xxx 的内部结构、等效电路与引脚排列

(a) 内部结构; (b) 等效电路; (c) 引脚排列

MCP41xxx数字电位器的引脚排列如图6-11(c)所示。其中:

- (1) $\overline{\text{CS}}$ 是片选输入端, 低电平有效。
- (2) SCK是串行数据输入的同步时钟。在数据准备好的情况下, SCK的下降沿同步输入

数据。

(3) SI是串行数据输入信号。在SCK的配合下, SI向器件输入数据。

(4) GND是参考地。

(5) PAO是数字电位器的一个固定端, 见图6-11(b)中的PAO位置。

(6) PWO是数字电位器的抽头滑动端, 见图6-11(b)中的PWO位置。

(7) PBO是数字电位器的一个固定端, 见图6-11(b)中的PBO位置。

(8) Vcc是电源输入端。

另外, MCP42xxx系列是一个双数字电位器, 其原理和电特性与MCP41xxx是一致的, 只是引脚有14脚(SOP封装)。

3. MCP41xxx的操作

MCP41xxx或MCP42xxx的操作是通过一个命令字节完成的。该命令字节的格式如图6-12所示。

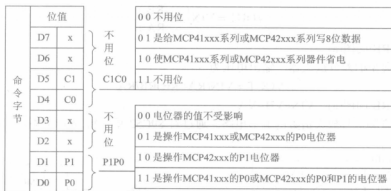


图 6-12 一个命令字节的格式

由图6-12可知, 一个字节命令实际上只对C1、C0位(功能选择)和P1、P0位(电位器选择)进行设置即可。对于MCP41xxx系列器件来说, 只有一个电位器P0, 而MCP42xxx系列器件才有P1与P0两个电位器。

MCP41xxx的工作时序如图6-13所示。

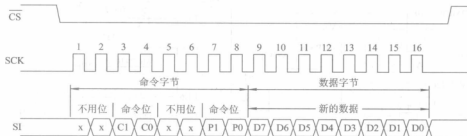


图 6-13 MCP41xxx 的工作时序

由于该电位器的数字调节范围是0~256, 因此数字每增加一位(或减少一位)电位器的值就会成比例地增加(或减少)。为了确定电位器的当前“位置”, 每次上电或重新复位“数据

字节”的数据被初始化为 80H(即电位器的滑动端处在中心位置)。这样就可通过外控的 MCU 实现对 MCP41xxx 或 MCP42xxx 的“精确位置”调节。

6.4.2 应用电路与编程

1. 应用电路

图 6-14 是 MCP41010 和 MCP42010 的应用的几种接法。其中, 图(a)是典型的反相放大器。输出电压为

$$V_{OUT} = -V_{IN} \left(\frac{R_B}{R_A} \right) + V_{REF} \left(1 + \frac{R_B}{R_A} \right)$$

式中, $R_A = [RAB(256 - DN)]/256$, $R_B = RAB \cdot DN/256$, RAB 是总电阻, DN 是写入的数字 (0~255)。

图(b)是典型的同相放大器。输出电压为

$$V_{OUT} = V_{IN} \left(1 + \frac{R_B}{R_A} \right)$$

式中, R_A 、 R_B 同图(a)。

图(c)是典型的差分放大器。输出电压为

$$V_{OUT} = V_{IN}(R_A - R_B)R_B/R_A$$

式中, R_A 、 R_B 同图(a), $V_{IN} = V_B - V_A$ 。

图(d)是音频信号放大倍数的调节电路。通过对 89C52 单片机 I/O 口编程可实现喇叭音量的控制。

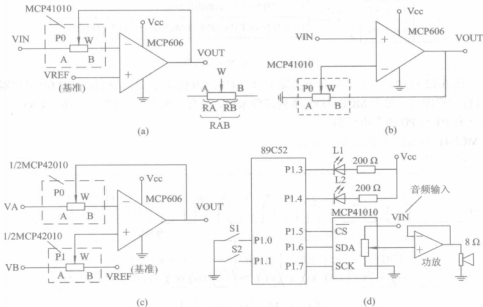


图 6-14 几种基本应用

(a) 反相放大器; (b) 同相放大器; (c) 差分放大器; (d) 音量控制电路

2. 编程方法

MCP41xxx 系列器件是 SPI 总线接口, 它的内部无非易失性存储器, 只有 16 位的数据锁存器。其中的 8 位数据正好控制 256 个电阻滑动点。也就是说, 数字量 0~255 对应 0~255 个电阻位置。为了编程清楚电位器的“位置点”, 该器件在上电时已将内部初始化成 80H(即 128), 这个值正好是电阻位置的“中间点”(即总电阻值的一半)。所以在编程时, 可以设一个字符型变量, 每次开机时可以将该变量确定为 80H, 每操作一次 MCP41xxx 器件, 该变量相应增加同样的值, 即可解决任意电阻位置的问题。

结合图 6-14(d)用 C51 实现的相关程序如下:

```
#include <reg52.h>

sbit S_led=P1^3;          /*定义上升指示灯引脚*/
sbit D_led=P1^4;          /*定义下降指示灯引脚*/
sbit CS=P1^5;             /*定义器件片选引脚*/
sbit SDA=P1^6;            /*定义数据引脚*/
sbit SCK=P1^7;            /*定义时钟引脚*/
sbit S1=P1^0;             /*定义向上的键引脚*/
sbit S2=P1^1;            /*定义向下的键引脚*/

#define uchar unsigned char
#define DATA_CD 0x11     /*定义 P0 电位器送数据指令*/
#define DOWN_CD 0x41     /*定义 P0 电位器掉电模式指令*/
uchar count=0;           /*定义的计数变量*/
void delay(unsigned int k) /*短延时函数, k 是时常数*/
{
    unsigned int i;
    for(i=0; i<k; i++);
}

/*向 MCP41xxx 写数据, 其中, cd 是命令, dat_a 是数据*/
void wire_MCP_16(uchar cd, uchar dat_a)
{
    uchar j;
    unsigned int x;        /*定义整型变量*/
    CS=1; SDA=0; SCK=1;   /*初始化*/
    x=(unsigned int)cd*256+(unsigned int)dat_a; /*将两个 8 位变成 16 位数据*/
    CS=0; delay(0x10); SCK=0;
    for(j=0; j<16; j++)
    {
        SCK=0; delay(0x5); /*SCK=0, 并延时*/
        if((x && 0x8000)==1) SDA=1;
        else SDA=0;
        x=x<<1;
        SCK=1; delay(0x5); /*SCK=1, 并延时*/
    }
}
```

```

    }
    CS=1; /*结束*/
}

/* MCP41xxx 写命令*/
void write_MCP_8(uchar cd)
{
    uchar j,x;
    CS=1; SDA=0; SCK=1; /*初始化*/
    x=cd; /*将一个 8 位命令写入 MCP41xxx*/
    CS=0; delay(0x10); SCK=0;
    for(j=0; j<8; j++)
    {
        SCK=0; delay(0x5); /*SCK=0, 并延时*/
        if((x && 0x80)==1) SDA=1;
        else SDA=0;
        x=x<<1;
        SCK=1; delay(0x5); /*SCK=1, 并延时*/
    }
    CS=1; /*结束*/
}

/*向下滑动一个数字*/
void down(uchar data_p0)
{
    S_led=1; D_led=0; /*向下灯点亮*/
    if(data_p0 <=0) data_p0=0; /*保证数据在最下端固定*/
    write_MCP_16(DATA_CD,data_p0); /*向下移动*/
}

/*向上滑动一个数字*/
void up(uchar data_p0)
{
    D_led=1; S_led=0; /*向上灯点亮*/
    if(data_p0 >=0xff) data_p0=0xff; /*保证数据在最上端固定*/
    write_MCP_16(DATA_CD,data_p0); /*向上移动*/
}

void main(void) /*主函数部分, 程序中有按键演示升和降的过程*/
{
    up(0x90);
    count=0x80; /*将计数器设成 0x80, 保持与开机(上电)的值一致*/
    write_MCP_8(DOWN_CD); /*进入省电模式*/
    while(1)

```

```

    {
        if(ku==0) /*向上移动一个位置*/
        {
            count++; up(count);
            delay(0x100);
        }
        if(kd==0) /*向下移动位置*/
        {
            count--; down(count);
            delay(0x100);
        }
    }
}
}

```

6.5 X9111 低功耗 1024 抽头数字电位器

6.5.1 硬件与功能描述

X9111 器件是具有 1024 个抽头的数字电位器(XDCP)。X9111 内部集成有 1023 个电阻单元、控制滑动位置的计数寄存器、记忆位置的非易失性寄存器和多路电子开关。

该器件可作为标准的三端口电位器(或二端口电位器),广泛应用于电压放大器的增益调节、可编程基准电源比较、精密电压检测、音频电路控制、电压放大器的误差调整、滤波参数和可编程信号发生器的控制等系统中。

1. 主要性能特点

- (1) 总电阻为 100 k Ω ;
- (2) 有 1024 个电阻抽头, 10 位的数字分辨率;
- (3) SPI 串行接口;
- (4) 4 个非易失性数字寄存器, 能随时记忆滑动点位置;
- (5) 每位可允许 10 000 次数据擦写;
- (6) 数据保存至少 100 年;
- (7) 上电后可装回上次断电前的位置点;
- (8) 宽电压 CMOS 技术, 最大静态电流为 3 μ A;
- (9) 电源范围为 2.7~5.5 V;
- (10) 器件工作温度为 -40~+85 $^{\circ}$ C。

2. 内部结构与引脚说明

X9111 数字电位器的内部结构如图 6-15(a)所示。

X9111 数字电位器的引脚采用 TSSOP-14 脚封装, 排列见图 6-15(c)。其中:

- (1) SO 是串行数据输出端, 在时钟的下降沿将数据输出。
- (2) SI 是串行数据输入端。SI 在同步时钟的下降沿将所有命令、地址和数据从该脚输入到 X9111 中。
- (3) A0、A1 是器件从地址输入端。从地址的串行数据流必须与输入地址相匹配才能开

始与该器件的通信。

(4) SCK 是串行接口的时钟输入端,用于同步数据输入/输出。

(5) CS 是片选输入端,低电平有效。

(6) WP 是硬件写输入保护端,低电平有效。

(7) HOLD 是数据保持端。当 $\overline{\text{HOLD}}=0$ 时,被选中的器件的串行数据暂停。

(8) RH、RL 和 RW 分别表示数字电位器的三个端口,即两个固定端口和一个滑动端口,如图 6-15(b)所示。

(9) Vcc、GND 是电源端和参考地。

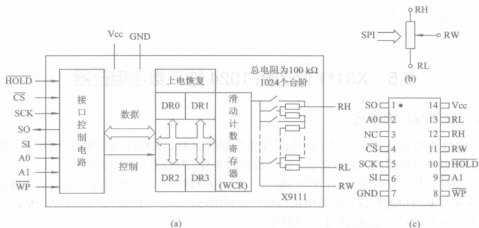


图 6-15 X9111 的内部结构、等效电位器与引脚排列

(a) 内部结构; (b) 等效电位器; (c) 引脚排列

3. 器件的操作

X9111 数字电位器滑动点的位置直接由“滑动计数寄存器(WCR)”的值确定。当 WCR 为 000H 时,滑动点移到 RL 端($\text{RW}=\text{RL}$);当 WCR 为 3FFH 时,滑动点移到 RH 端($\text{RW}=\text{RH}$)。WCR 的任意计数值通过 SPI 接口输入。WCR 的值可以存入非易失性数字寄存器 DR3~DR0(每个寄存器为 10 位)中,在上电时,器件内部会自动将 DR0 的值装入 WCR 中,以防止电位器跳变。X9111 的状态寄存器只有一位 WIP 是只读位,该位是写状态指示位, WIP=0 表示写结束, WIP=1 表示写正在进行。

WCR 计数寄存器、DR3~DR0 数据寄存器的格式如表 6-4 所示。

表 6-4 WCR、DR3~DR0 的格式

D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
WCR 计数寄存器									
WCR9	WCR8	WCR7	WCR6	WCR5	WCR4	WCR3	WCR2	WCR1	WCR0
DR3~DR0 非易失性(NV)数据寄存器									
NV9	NV8	NV7	NV6	NV5	NV4	NV3	NV2	NV1	NV0

1) 器件的标识

X9111 的标识占一个字节, 由身份号(ID)、器件地址和读/写位组成, 见表 6-5。其中, ID3~ID0(0101)是器件的身份标志, A1、A0(11、10、01、00)是器件的从地址, 当外引脚 A1、A0 的状态与内部 A1、A0 的从地址值相匹配时, 才能对该器件操作。R/W 是读/写位, $R/\overline{W}=1$ 是读, $R/\overline{W}=0$ 是写。

表 6-5 器件的标识

D7	D6	D5	D4	D3	D2	D1	D0
器件的身份号				0	地址		读/写
ID3	ID2	ID1	ID0		A1	A0	R/W
0	1	0	1		11、10、01、00		1/0

2) 器件的指令

X9111 的指令占一个字节, 由指令码(I2、I1、I0)和寄存器选择位(PB、PA)组成, 见表 6-6。其中, PB、PA = 00 选择 DR0; PB、PA = 01 选择 DR1; PB、PA = 10 选择 DR2; PB、PA = 11 选择 DR3。

表 6-6 器件的指令

D7	D6	D5	D4	D3	D2	D1	D0
指令码(见表 6-7)			0	寄存器选择		0	0
I2	I1	I0		PB	PA		

在表 6-7 所示的指令中有 5 个指令长度是 4 个字节长。这 5 个指令分别是: 读当前滑动计数寄存器(WCR)内容、写(改变)当前滑动计数寄存器、读所选择的(DR3~DR0 中的一个)数据寄存器、写所选择的(DR3~DR0 中的一个)数据寄存器和读状态位的指令。

表 6-7 指令约定

指令功能	R/\overline{W}	指令码								含 义
		I2	I1	I0	0	PB	PA	0	0	
读计数寄存器	1	1	0	0	0	0	0	0	0	读滑动计数器的内容
写计数寄存器	0	1	0	1	0	0	0	0	0	写滑动计数器的值
读数据寄存器	1	1	0	1	0	1/0	1/0	0	0	读PB、PA选择的寄存器内容
写数据寄存器	0	1	1	0	0	1/0	1/0	0	0	写PB、PA选择的寄存器内容
XFR到WCR	1	1	1	0	0	1/0	1/0	0	0	将PB、PA选择的XFR传WCR
WCR到XFR	0	1	1	1	0	1/0	1/0	0	0	WCR传PB、PA选择的XFR
读状态位	1	0	1	0	0	0	0	0	1	读状态寄存器的WIP位

3) 操作时序

2 个字节长度的指令时序如图 6-16 所示。2 个字节指令分别用来将 PB、PA 选择的寄存

器内容读到计数寄存器中和将计数寄存器的值写到由 PB、PA 选择的非易失性寄存器中。

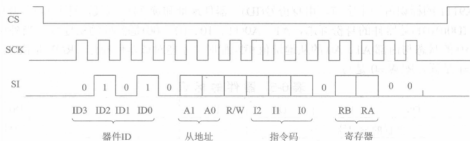


图 6-16 2 个字节长度的指令时序

四字节指令分为读/写计数寄存器(或读/写非易失寄存器)时序(见图 6-17)和读状态位时序(见图 6-18)。

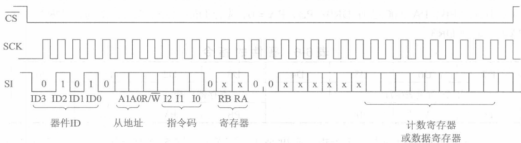


图 6-17 读/写数据四字节时序

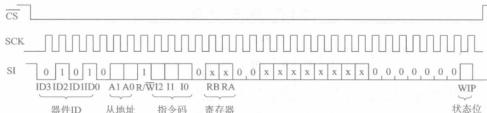


图 6-18 读 WIP 位四字节时序

6.5.2 应用电路与编程

1. 应用电路

X9111 有 1024 个台阶,每一台阶电阻变化约 $0.087 \text{ k}\Omega$,分辨精度较高,应用范围很广。图 6-19 是由 X9111 构成的几种放大器应用电路。其中,图(a)是基本信号衰减器;图(b)是可控一阶高通滤波器,截止频率为 $1/(2\pi \times C1 \times R3)$;图(c)是 L-R 等效电路;图(d)是最基本的同相放大器。通过对各自 X9111 的电阻操作可实现程控的目的。

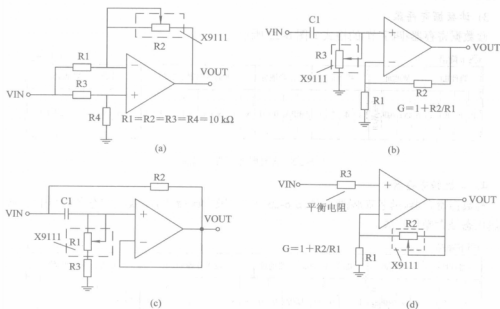


图 6-19 由 X9111 构成的几种放大器应用电路

(a) 衰减器; (b) 滤波器; (c) L-R 等效电路; (d) 同相放大器

2. 编程方法

根据 X9111 的工作原理和 SPI 接口协议, 按照表 6-7 所示的指令格式和二字节、四字节时序用 C51 实现编程并不难。在写操作时, 要查询状态位 WIP。当 WIP = 1 时, 表示正在操作; 当 WIP = 0 时, 表示该操作完成。

1) 读计数寄存器

读计数寄存器(四字节)的格式如图 6-20 所示。

CS下降沿										CS上升沿																																				
器件ID				从地址		指令码		寄存器地址		滑动端高2位								滑动端低8位																												
0	1	0	1	0	A1	A0	R/W=1	1	0	0	0	0	0	0	0	x	x	x	x	x	W	C	R	9	8	W	C	R	7	6	W	C	R	5	4	W	C	R	3	2	1	0	W	C	R	R

图 6-20 读 WCR 数据流

2) 写计数寄存器

写计数寄存器(四字节)的格式如图 6-21 所示。

CS 下降沿										CS 上升沿																
器件ID	从地址	指令码	寄存器地址	滑动端高2位								滑动端低8位														
0	1	0	1	0	A1	A0	R/W=0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图 6-21 写 WCR 数据流

3) 读数据寄存器

读数据寄存器(四字节)的格式如图 6-22 所示。

CS下降沿										CS上升沿																					
器件ID		从地址		指令码		寄存器地址		所选寄存器高2位								所选寄存器低8位															
0	1	0	1	0	A1	A0	$R/\overline{W}=1$	1	0	1	0	PB	PA	0	0	x	x	x	x	x	x	9	8	7	6	5	4	3	2	1	0

图 6-22 读数据寄存器数据流

4) 写数据寄存器

写数据寄存器(四字节)的格式如图 6-23 所示。操作该指令后,要查状态寄存器中的 WIP 位的状态是否操作完成。

CS下降沿														CS上升沿																
器件ID		从地址		指令码		寄存器地址		所选寄存器高2位								所选寄存器低8位														
0	1	0	1	0	A1	A0	R/W=0	1	1	0	0	PB	PA	0	0	x	x	x	x	x	9	8	7	6	5	4	3	2	1	0

图 6-23 写数据寄存器数据流

5) 传送数据寄存器(DR)到 WCR

传送数据寄存器(DR)到计数寄存器(WCR)的格式如图 6-24 所示。

CS下降沿										CS上升沿									
器件ID				从地址				指令码				寄存器地址							
0	1	0	1	0	A1	A0	R/W=1	1	1	0	0	PB	PA	0	0				

图 6-24 传送 DR 到 WCR 数据流

6) 传计数寄存器(WCR)到 DR

传送计数寄存器(WCR)到数据寄存器(DR)的格式如图 6-25 所示。操作该指令后,要查状态寄存器的 WIP 位的状态是否操作完成。

CS下降沿										CS上升沿									
器件ID				从地址				指令码				寄存器地址							
0	1	0	1	0	A1	A0	R/W=0	1	1	1	0	PB	PA	0	0				

图 6-25 传送 WCR 到 DR 数据流

7) 读状态寄存器的 WIP 位

读状态寄存器 WIP 位的格式如图 6-26 所示。

CS下降沿										CS上升沿																		
器件ID				从地址		指令码		寄存器地址		状态数据					状态数据													
0	1	0	1	0	A1 A0	R/W=1	0	1	0	0	0	0	1	x	x	x	x	x	x	0	0	0	0	0	0	0	0	WIP

图 6-26 读状态寄存器 WIP 位

8) 相关的 C51 源程序

相关的 C51 源程序如下:

```
#include <reg52.h>
/*MCU 采用 51 单片机, 引脚用到 P1.0~P1.5*/

sbit CS = P1^0;          /*定义器件片选引脚*/
sbit DI = P1^1;          /*定义数据送入引脚*/
sbit SO = P1^2;          /*定义数据输出引脚*/
sbit SCK = P1^3;         /*定义时钟引脚*/
sbit WP = P1^4;          /*写保护引脚*/
sbit HOLD = P1^5;        /*数据保护输入引脚*/

#define uchar unsigned char
#define uint unsigned int

#define W_ID 0x50         /*定义 0 器件写标识字节*/
#define R_ID 0x51         /*定义 0 器件读标识字节*/
#define R_WCR 0x80        /*定义 0 器件读 WCR 数据指令字节*/
#define W_WCR 0xA0        /*定义 0 器件写 WCR 数据指令字节*/
#define R_DR0 0xA0        /*定义 0 器件读 DR0 数据指令字节*/
#define W_DR0 0xC0        /*定义 0 器件写 DR0 数据指令字节*/
#define M_DR0_WCR 0xC0    /*定义 0 器件 DR0 到 WCR 指令*/
#define M_WCR_DR0 0xE0    /*定义 0 器件 WCR 到 DR0 指令*/
#define R_SR 0x41         /*定义 0 器件读状态寄存器指令*/

uint count=0;            /*定义的计数变量*/
void delay(uint k)        /*短延时函数, k 是时常数*/
{
    uint i;
    for (i=0; i<k; i++);
}

/*向 X9111 写二字节数据, 其中, id 是标识, cd 是命令*/
void write_X9111_2(uchar id, uchar cd)
{
    uchar j;
    uint x;                /*定义整型变量*/
```

```

CS=1; DI=0; SCK=1; SO=1;    /*初始化*/
x=(uint)id*256+(uint)cd;      /*将两个8位数据拼成16位指令*/
CS=0;                          /*CS产生下降沿*/
delay(0x10);
for(j=0; j<16; j++)
{
    if((x & 0x8000)==1) DI=1;
    else DI=0;
    x=x<<1;
    SCK=0; delay(0x5);        /*SCK产生下降沿,并延时*/
    SCK=1; delay(0x5);        /*SCK=1*/
}
CS=1;                          /*结束*/
}

/*向X9111写四字节数据,其中,id是标识,cd是命令,y是要写的数据*/
void wire_X9111_4(uchar id,uchar cd,uint y)
{
    uchar j;
    uint x;
    CS=1; DI=0; SCK=1; SO=1;    /*初始化*/
    x=(uint)id*256+(uint)cd;      /*将两个8位数据拼成16位指令*/
    CS=0;                          /*CS产生下降沿*/
    delay(0x10);                  /*延时*/
    for(j=0; j<16; j++)
    {
        if((x & 0x8000)==1) DI=1;
        else DI=0;
        x=x<<1;
        SCK=0; delay(0x5);        /*SCK产生下降沿,并延时*/
        SCK=1; delay(0x5);        /*SCK=1*/
    }
    x=y&0x3ff;                  /*10位数据*/
    for(j=0; j<16; j++)          /*送数据*/
    {
        if((x & 0x8000)==1) DI=1;
        else DI=0;
        x=x<<1;
        SCK=0; delay(0x5);        /*SCK产生下降沿,并延时*/
        SCK=1; delay(0x5);        /*SCK=1*/
    }
}

```

```

    }
    CS=1;          /*结束*/
}

/*读一个 16 位数据, id 是标识, cd 是指令, 返回一个整型数据*/
uint read_X9111_4(uchar id,uchar cd)
{
    uchar j;
    uint x;
    CS=1; DI=0; SCK=1; SO=1; /*初始化*/
    x=(uint)id*256+(uint)cd; /*将两个 8 位数据拼成 16 位指令*/
    CS=0; /*CS 产生下降沿*/
    delay(0x10); /*延时*/
    for(j=0; j<16; j++)
    {
        if((x & 0x8000) == 1) DI=1;
        else DI=0;
        x=x<<1;
        SCK=0; delay(0x5); /*SCK 产生下降沿, 并延时*/
        SCK=1; delay(0x5); /*SCK=1*/
    }
    x=0;
    for(j=0; j<16; j++) /*读数据*/
    {
        x=x<<1;
        SCK=0; delay(0x5);
        if(SO == 1) x=x|0x0001;
        else x=x|0x0000;
        SCK=1; delay(0x5);
    }
    CS=1; /*结束*/
    return x;
}

/*读状态寄存器的 WIP*/
void read_WIP(void)
{
    uint wip=1;
    while(wip==1) /*等待 WIP 变 0*/
    {
        wip=read_X9111_4(R_ID,R_SR); /*读 WIP 位*/
    }
}

```

```

        wip=wip & 0x0001;
    }

}

/*读 WCR 的数据*/
uint read_wcr(void)
{
    uint x;
    x=read_X9111_4(R_ID,R_WCR);    /*读 WCR 数据*/
    return x;
}

/*读 DR0 的数据*/
uint read_dr0(void)
{
    uint x;
    x=read_X9111_4(R_ID,R_DR0);    /*读 WCR 数据*/
    return x;
}

/*写一个数据到计数寄存器(WCR)*/
void write_wcr(uint x)
{
    uint y;
    if (x>=0x3ff) y=0x3ff;
    else y=x;
    write_X9111_4(W_ID,W_WCR,y);    /*写一个数据到 WCR 控制滑动点*/
    read_WIP();
}

/*将 DR0 数据送回计数寄存器(WCR)*/
void mov_DR0_wcr( void )
{
    write_X9111_2(R_ID,M_DR0_WCR); /*将 DR0 送到 WCR*/
    read_WIP();                     /*判断 WIP 是否等于 1*/
}

/*将 WCR 数据送回 DR0*/
void mov_WCR_dr0( void )
{
    write_X9111_2(W_ID,M_WCR_DR0);
    read_WIP();                     /*判断 WIP 是否等于 1*/
}

void main(void)
/*主函数部分演示电阻增大的过程*/

```

```

uint j;
HOLD=1; WP=1; /*初始化*/
mov_DR0_wcr(); /*恢复 WCR 的数据*/
count=0; /*将计数器设成 0*/
for (j=0; j<0x3ff; j++)
{
    write_wcr(j); /*调节数字电位器逐级增大*/
    delay(0x10); /*延时*/
}
mov_WCR_dr0(); /*将 WCR 的最后数据保存到 DR0 中*/
while(1);
}

```

6.6 CAT512x 系列 2 线接口数字电位器

6.6.1 硬件与功能描述

CAT5120/5121/5122 器件是具有 16 个抽头的线性数字电位器。内部集成有 15 个电阻单元、控制滑动位置的计数寄存器和多路电子开关。

CAT5120 是标准的三端口电位器, CAT5121 和 CAT5122 是标准的二端口电位器。它们可广泛应用于电压放大器的增益调节、可编程基准电源比较和音量控制等系统中。

1. 主要性能特点

- (1) 总电阻有 10 k Ω 、50 k Ω 和 100 k Ω ;
- (2) 0.3 μ A 的超低待机电流;
- (3) 电阻值切换时没有干扰;
- (4) 有 16 个电阻抽头, 4 位的数字分辨率;
- (5) 2 线串行接口, 5 脚或 6 脚封装;
- (6) 上电自动将滑动点恢复到标称电阻的中间位置;
- (7) 固定端热稳定度(典型值)为 $200 \times 10^{-6}/^{\circ}\text{C}$;
- (8) 相对端热稳定度(典型值)为 $5 \times 10^{-6}/^{\circ}\text{C}$;
- (9) 电源范围为 2.7~5.5 V;
- (10) 器件工作温度为 $-40 \sim +85^{\circ}\text{C}$ 。

2. 内部结构与引脚说明

CAT5120/5121/5122 数字电位器的内部结构如图 6-27(a)所示。图 6-27(b)是器件的引脚排列, 外形为 6 脚 SC-70 和 SOT-23 封装。其中:

- (1) Vcc 是电源引脚, 一般接+5 V 电源。
- (2) U/ $\overline{\text{D}}$ 是升/降控制输入脚。当 U/ $\overline{\text{D}}$ = 1 时, 在 $\overline{\text{CS}}$ 从高到低的跳变开始, U/ $\overline{\text{D}}$ 的变化将是电阻增加; 当 U/ $\overline{\text{D}}$ = 0 时, 在 $\overline{\text{CS}}$ 从高到低的跳变开始, U/ $\overline{\text{D}}$ 的变化将是电阻减少。

(3) $\overline{\text{CS}}$ 是片选输入端, 低电平有效。在 $\overline{\text{CS}}$ 从高到低的跳变开始, 是升还是降取决于 $\text{U}/\overline{\text{D}}$ 当时的状态。

(4) RL 是 CAT512x 的等效电阻低端。封装不一样, RL 的输出也有差别。

(5) RH 是 CAT512x 的电阻高端, 等效于机械电位器的固定端。封装不一样, RL 的输出也有差别。

(6) RW 是 CAT512x 的电阻的滑动端。封装不一样, RW 的输出也有差别。

(7) GND 是器件的参考地。

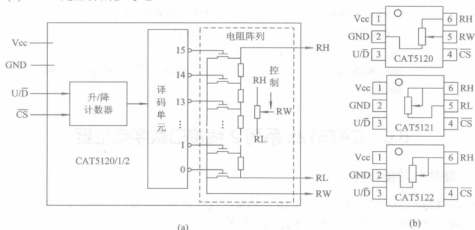


图 6-27 CAT5120/5121/5122 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

3. CAT512x 的操作

$\text{U}/\overline{\text{D}}$ 和 $\overline{\text{CS}}$ 两个输入信号直接控制电位器滑动点的位置。当 $\text{U}/\overline{\text{D}}$ 为高电平时, 若 $\overline{\text{CS}}$ 从高到低变化, 则器件将进入增加模式; 当 $\text{U}/\overline{\text{D}}$ 为低电平时, 若 $\overline{\text{CS}}$ 从高到低变化, 则器件将进入减小模式。一旦进入增加或减小模式, 器件将一直维持这个模式, 直到 $\overline{\text{CS}}$ 变高为止。在增加或减小模式下, $\text{U}/\overline{\text{D}}$ 每有一个由低到高的变化(上升沿), 电位器都将增加或减小一个台阶, 其时序如图 6-28 和图 6-29 所示。

为了确定电位器的位置, 每一次上电, 内部计数器都将复位, 滑动点处在总电阻的中间位置, 且默认为增加模式。

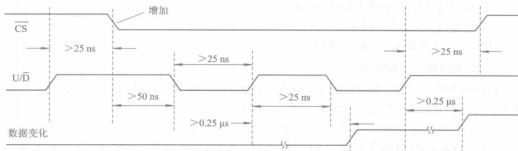


图 6-28 2 线接口增加模式时序

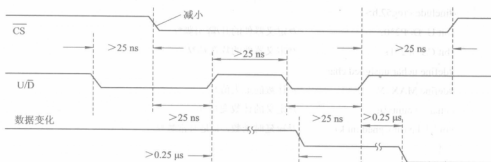


图 6-29 2 线接口减小模式时序

6.6.2 应用电路与编程

1. 应用电路

CAT5120/5121/5122 的几种应用电路如图 6-30 所示。其中,图(a)、(b)是两种 LCD 背光调整电路,图(c)、(d)是两种同相放大器电路。

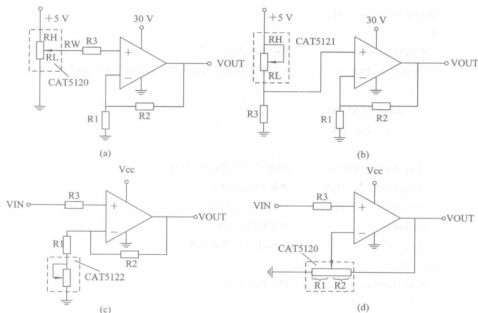


图 6-30 CAT512x 的几种应用电路

(a)、(b) 两种 LCD 背光调整电路; (c)、(d) 两种同相放大器

2. 编程方法

设单片机(STC89C52RD)的 P2.0、P2.1 引脚分别接 CAT512x 的 $\overline{U/D}$ 和 \overline{CS} , 用 C51 实现的源程序如下:


```

#include <reg52.h>

sbit U_D=P2^0;          /*定义器件的升/降引脚*/
sbit CS =P2^1;          /*定义器件的片选端*/

#define uchar unsigned char
#define MAX_N 15        /*计数的最大值*/
uchar count=0;          /*定义的计数变量*/
void delay(unsigned int k) /*短延时函数, k 是时常数*/
{
    unsigned int i;
    for(i=0; i<k; i++);
}

void CAT512Init(void)    /*初始化*/
{
    CS=1;                /*片选高*/
    U_D=1;               /*默认上升*/
}

/*滑动点位置增 1*/
void Increment_one(void)
{
    CAT512Init();        /*初始状态*/
    if(count>=MAX_N)     /*判断是否增加到最大值, 若是最大值, 则不增 1*/
    {
        count=MAX_N;
        goto quit_INC;
    }

    delay(0x10); CS=0;    /*CS 产生下降沿(U_D=1)*/
    delay(0x10); U_D=0;   /*变下降沿(0)*/
    delay(0x10); U_D=1;   /*变上升沿(1)*/
    count++;              /*计数器加 1*/
    delay(0x10);          /*延时, 电阻向上增 1 步*/

quit_INC:
    CAT512Init();        /*初始状态*/
}

/*滑动点位置减 1*/
void Decrement_one(void)
{
    CAT512Init();        /*初始状态*/
    if(count<=0)         /*判断是否减到 0, 若减到 0, 则不变*/
    {

```

```

count=0;
goto quit_DEC;
}
U_D=0; delay(0x10); /*置为减小模式*/
CS=0; delay(0x10); /*CS产生下降沿(U_D=1)*/
U_D=1; /*变上升沿(1)*/
count--; /*计数器减1*/
delay(0x10);
quit_DEC:
    CAT512Init(); /*初始状态*/
}
void main(void) /*主函数部分*/
{
    CAT512Init(); /*初始状态*/
    count=8; /*将计数器设为中间位置*/
    Increment_one(); /*电位器电阻加一步*/
    /*Decrement_one(); /*电位器电阻减一步*/
    delay_s(0x10); /*延时*/
    while(1);
}

```

6.7 CAT5112 带缓冲滑动的 32 抽头数字电位器

6.7.1 硬件与功能描述

CAT5112 是可编程数字电位器, 在电子系统中, 它可以完全取代机械电位器和微调电阻。CAT5112 包含一个连接到两个固定端和 32 抽头的串联电阻阵列。通过递增/递减计数器译码器可控制滑动点的位置, 滑动点由运算放大器缓冲。滑动计数位置由非易失性存储器保存, 不会在器件掉电时丢失, 可在器件上电后自动恢复。

CAT5112 带有缓冲的数字电位器, 当用作两端可变电阻时, 可用来控制、调节或校准模拟电路的特性和参数, 其应用非常广泛。

1. 主要性能特点

- (1) 总电阻有 10 k Ω 、50 k Ω 和 100 k Ω ;
- (2) 有 32 个电阻抽头, 带有缓冲器;
- (3) 电阻值切换时, 没有干扰;
- (4) 非易失性存储器存储滑动数据, 保存数据大于 100 年;
- (5) 递增/递减串行接口;
- (6) 电源范围为 2.5~6 V;
- (7) 器件工作温度为 0~+70 $^{\circ}\text{C}$ (民品), -40~+85 $^{\circ}\text{C}$ (工业级)。

2. 内部结构与引脚说明

CAT5112 数字电位器的内部结构与引脚排列如图 6-31 所示。

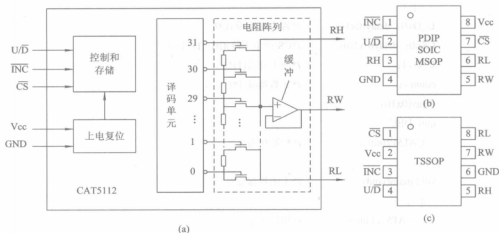


图 6-31 CAT5112 的内部结构与引脚排列

(a) 内部结构; (b)、(c) 引脚排列

器件的外形有 PDIP、SOIC、TSSOP 和 MSOP 几种封装。其中:

(1) \overline{INC} 是增加(或减小)输入脚, 下降沿触发。根据 U/\bar{D} 的输入状态可使滑动片向上或向下移动。

(2) U/\bar{D} 是升/降方向输入脚。当它为高且 \overline{CS} 为低时, \overline{INC} 脚上电平的负跳变将使滑动片向 RH 端移动一个增量; 当它为低且 \overline{CS} 为低时, \overline{INC} 脚上电平的负跳变将使滑动片向 RL 端移动一个增量。

(3) \overline{CS} 是片选端, 低电平有效。当它为高时, \overline{INC} 和 U/\bar{D} 的输入不会影响或改变滑动片的位置。

(4) RH 、 RL 是电位器的两个固定端(相当于机械电位器的两个固定端)。

(5) RW 是带缓冲的滑动端。它在电阻阵列上的位置受 \overline{INC} 、 U/\bar{D} 和 \overline{CS} 输入信号的控制。

(6) V_{cc} 、 GND 是电源和地。

3. 器件的操作

CAT5112 的操作类似于一个数控电位器, 它的 RH 端和 RL 端等效于机械电位器的高端和低端, RW 等效于机械电位器的滑动片。

当 \overline{CS} 端置为低时, CAT5112 就被选中, 开始响应 U/\bar{D} 和 \overline{INC} 的输入。 \overline{INC} 电平的负跳变会使滑动片上移或者下移(取决于 U/\bar{D} 输入和 5 位计数器的状态)。当滑动片位于任何一个固定端时, 它的操作就和机械电位器一样, 移动时不会超过末端位置。

当 \overline{CS} 变为高而 \overline{INC} 输入也保持高电平时, 计数器的值就保存到非易失性存储器中。当器件断电后, 最后存储的滑动片计数器位置仍然保存在非易失性存储器中; 当电源恢复后, 存储器中的值重新调入计数器。其操作模式如表 6-8 所示。

表 6-8 CAT5112 的操作模式

CS 状态	INC 状态	U/D 状态	模 式
低电平	由高向低(下降沿)	高电平	滑动向RH方向移动
低电平	由高向低(下降沿)	低电平	滑动向RL方向移动
由低向高(上升沿)	高电平	高电平	保存滑动片位置
由低向高(上升沿)	低电平	高电平	不存储, 返回待机状态
高电平	高电平	高电平	进入待机状态

6.7.2 应用电路与编程

CAT5112 类似于机械电位器, 但因滑动端加有运放缓冲, 故输出阻抗较低, 容易与负载匹配。CAT5112 比较适合分压、偏移、增益和零点调整等应用(参考图 6-14 和图 6-30)。

通过C51实现CAT5112电阻增加或减小的相关程序如下:

```
#include <reg52.h>

sbit U_D=P2^0;          /*定义器件的升/降引脚*/
sbit CS=P2^1;           /*定义器件的片选引脚*/
sbit INC=P2^2;          /*定义器件的增加/减小引脚*/

void delay(unsigned int k) /*短延时函数, k 是时常数*/
{
    unsigned int i;
    for(i=0; i<k; i++);
}

void CAT5112Init(void) /*初始化*/
{
    CS=1;               /*片选置高*/
    U_D=1; INC=1;       /*默认增加*/
}

/*滑动点位置增 1 或减 1, j = 0 时减小, j = 1 时增加*/
void INC_DEC_one(unsigned char j)
{
    CAT5112Init();      /*初始状态*/
    if(j==0)            /*向小调整*/
    {
        CS=0; U_D=0;
        delay(5);
        INC=0; delay(0x5);
    } /*向下调一步*/
    else if(j==1)       /*向大调整*/
    {
        CS=0; U_D=1;
        delay(5);
    }
}
```

```

    INC=0; delay(5); /*向上调一步*/
}
INC=1; U_D=1; delay(5); /*为存储做准备*/
CS=1; delay(10); /*将数据存储到 NRAM 中*/
}

void main(void) /*测试主函数*/
{
    CAT5112Init(); /*初始状态*/
    INC_DEC_one(1); /*电位器电阻加一步*/
    delay(10);
    while(1);
}

```

6.8 CAT5111 带缓冲滑动的 100 抽头数字电位器

6.8.1 硬件与功能描述

CAT5111 是一个可编程的 100 抽头式数字电位器。在电子系统中，可以完全取代机械电位器和微调电阻。CAT5111 内部包含有滑动计数器、非易失性存储器、电阻阵列、多路电子开关和输出缓冲器。滑动片的设置保存在非易失性存储器中，不会在器件掉电时丢失，可在器件上电后自动恢复。

CAT5111 带有缓冲的数字电位器，可用于产品自动校准、远程控制调节、偏移、增益和零点控制、对比度、亮度和音量控制、电机反馈系统控制和可编程模拟电路等系统。

1. 主要性能特点

- (1) 总电阻有 10 k Ω 、50 k Ω 和 100 k Ω ;
- (2) 有 100 个电阻抽头，带有缓冲器;
- (3) 电阻值切换时没有干扰;
- (4) 非易失性存储器存储滑动数据，保存数据大于 100 年;
- (5) 递增/递减串行接口;
- (6) 电源范围为 2.5~6 V;
- (7) 器件工作温度为 0~+70 $^{\circ}\text{C}$ (民品)，-40~+85 $^{\circ}\text{C}$ (工业级)。

2. 内部结构与引脚说明

CAT5111 数字电位器的内部结构如图 6-32(a)所示。该器件含有 100 个可用的抽头，RH 和 RL 端之间串联了 99 个电阻单元。滑动片端 RW 和 100 个抽头其中一个相连，由三个输入端 $\overline{\text{INC}}$ 、 U/D 和 $\overline{\text{CS}}$ 控制一个 7 位的递增/递减计数器，它的输出被译码后用来选择 RW 的位置。通过 $\overline{\text{INC}}$ 和 $\overline{\text{CS}}$ 输入端可将选择的滑动片位置存放到非易失性存储器中。

CAT5111 器件的外形有 PDIP、SOIC、TSSOP 和 MSOP 几种封装，引脚排列同图 6-31(b)。其中：

(1) $\overline{\text{INC}}$ 是增加(或减小)控制输入脚,下降沿触发。根据 $\text{U}/\overline{\text{D}}$ 的输入状态使滑动片向上或向下移动。

(2) $\text{U}/\overline{\text{D}}$ 是升/降方向控制输入脚。当 $\text{U}/\overline{\text{D}} = 1$, $\overline{\text{CS}} = 0$ 时, $\overline{\text{INC}}$ 脚上电平的正跳变将使滑动片向 RH 端移动一个增量;当 $\text{U}/\overline{\text{D}} = 0$, $\overline{\text{CS}} = 0$ 时, $\overline{\text{INC}}$ 脚上电平的正跳变将使滑动片向 RL 端移动一个增量。

(3) $\overline{\text{CS}}$ 是片选端,低电平有效。

(4) RH、RL 是电位器的两个固定端(相当于机械电位器的两个固定端)。

(5) RW 是带缓冲的滑动端。它在电阻阵列上的位置受 $\overline{\text{INC}}$ 、 $\text{U}/\overline{\text{D}}$ 和 $\overline{\text{CS}}$ 输入信号的控制。

(6) V_{CC} 、GND 是电源和地。

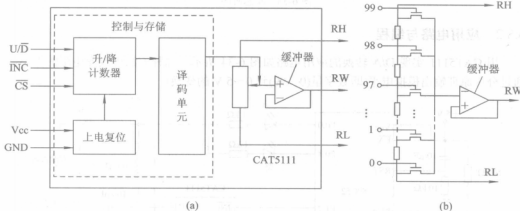


图 6-32 CAT5111 的内部结构和内部开关原理电路

(a) 内部结构; (b) 内部开关原理电路

3. 器件的操作

CAT5111 的操作类似于一个数字控制的电位器,它的 RH 端和 RL 端等效于机械电位器的高端和低端, RW 等效于机械电位器的滑动片。

4. 器件的操作

对 CAT5111 的操作,实际上是对接口信号的操作。当 $\overline{\text{CS}}$ 端置为低时, CAT5111 就被选中,开始响应 $\text{U}/\overline{\text{D}}$ 和 $\overline{\text{INC}}$ 的输入。 $\overline{\text{INC}}$ 的下降沿(负跳变)会使滑动片上移或者下移(取决于 $\text{U}/\overline{\text{D}}$ 输入和 7 位计数器的状态)。当滑动片位于任何一个固定端时,它的操作就和机械电位器一样,移动时不会超过末端位置。

当 $\overline{\text{CS}}$ 变为高而 $\overline{\text{INC}}$ 输入也保持高电平时,计数器的值就保存到非易失性存储器中。在器件断电后,最后存储的滑动片计数器位置仍然保存在非易失性存储器中。当电源恢复后,存储器中的值重新调入计数器。其操作模式如表 6-8 所示。

操作(动态)时序如图 6-33 所示。

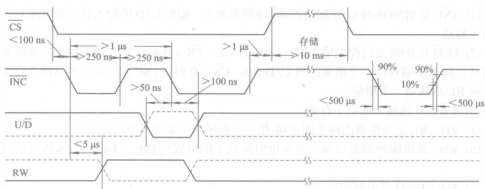


图 6-33 动态时序

6.8.2 应用电路与编程

用 CAT5111 实现 D/A 转换的应用电路如图 6-34 所示。这种方法可改变电阻的大小, 通过分压原理输出模拟电平(即数字量(0~99)到 0~5 V 的变换)。

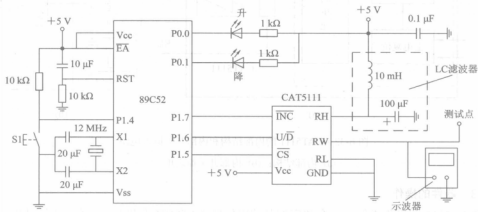


图 6-34 CAT5111 的应用电路

根据图 6-34 所示的连接方法, 要从 CAT5111 的 RW 端输出三角波, 则 C51 程序如下:

```
#include <reg52.h>
sbit U_D=P1^6; /*定义器件的升/降引脚*/
sbit CS =P1^5; /*定义器件的片选引脚*/
sbit INC=P1^7; /*定义器件的增加/减小引脚*/
sbit LED1=P0^0; /*定义器件升 LED1 灯引脚*/
sbit LED2=P0^1; /*定义器件降 LED2 灯引脚*/
void delay(unsigned int k) /*短延时函数, k 是时常数*/
{
    unsigned int i;
```

```

    for(i=0; i<k; i++){
    }
void CAT5111Init(void) /*初始化*/
{ CS=1; /*片选置高*/
  U_D=1; INC=1; /*默认增加*/
}
/*滑动点位置增1 函数*/
void INC_one(void)
{
    CAT5111Init(); /*初始状态*/
    LED1=0; LED2=1; /*升灯亮*/
    U_D=1; CS=0; delay(2); /*向增大方向移动一步*/
    INC=0; delay(2); /*INC 产生下降沿*/
    CS=1; delay(2); /*不存储*/
    INC=1;
}
/*滑动点位置减1 函数*/
void DEC_one(void)
{
    CAT5111Init(); /*初始状态*/
    LED1=1; LED2=0; /*降灯亮*/
    U_D=0; CS=0; delay(2); /*向减少方向移动一步*/
    INC=0; delay(2); /*INC 产生下降沿*/
    CS=1; delay(2); /*不存储*/
    INC=1;
}
void main(void) /*三角波产生函数*/
{
    unsigned char i;
    CAT5111Init(); /*初始状态*/
    for (i=0; i<0x63; i++) /*开机后将电位器调到 0*/
        DEC_one(); /*向小方向调*/
    while(1) /*产生三角波形*/
    {
        for(i=0; i<0x50; i++)
            INC_one(); /*从 0 往大增加, 电阻越大, 输出电压越高*/
        for(i=0; i<0x50; i++)

```



```
DEC_one(); /*从最大往小调,电阻越小,输出电压越小*/
```

6.9 X90100 非易失性可编程电容器

6.9.1 硬件与功能描述

X90100 是非易失性可编程电容器。该器件通过 3 线数字接口对内部电容进行设置,且保存在 E^2 PROM 中,输入由施密特触发器和上拉电阻来保证。在单端模式中有 32 个可编程电容值可供选择,范围是 7.5~14.5 pF(每步 0.23 pF)。绝缘体高度稳定,电容呈现一个非常低的电压系数。

X90100 可广泛应用于低成本再生接收器微调、低成本低温漂移的振荡器、车库开门器、无匙入口、工业无线控制、电容性传感器调整等系统。

1. 主要性能特点

- (1) 可编程电容,范围是 7.5~14.5 pF,每步 0.23 pF;
- (2) 有 32 个编程点,5 位数字分辨率;
- (3) 带有可编程微调码的非易失性 E^2 PROM 存储器;
- (4) 上电时自动设定可调电容;
- (5) 高性能可编程电容,线性误差小于 0.5LSB;
- (6) 调整最大递增量仅需 5 μ s;
- (7) 最小擦写次数为 100 000,数据保存期大于 100 年;
- (8) 电源范围为 2.7~5.5 V;
- (9) 器件工作温度为 -40~+85℃。

2. 内部结构与引脚说明

X90100 由高精度电容器、滑动控制计数器、多路电子开关和电源复位等电路组成,如图 6-35(a)所示。输入控制部分的工作就像一个升/降计数器。这个计数器的输出被译码而接通一个电子开关,以便把内部 31 个独立的电容连接到输出端。在特定条件下,计数器的内容可以存储在非易失性存储器中。当器件断电后,最后存储的计数器值在上电时会重新复制到滑动计数器中以保持原来的状态。

当电容器位于阵列范围中的任何一端(就像机械电容器的一个固定端)时,不会移动到超出终端位置,即当计数器达到一个极端时不会循环,除非反方向调整位置。

X90100 外形采用 8 脚 MSOP 封装,见图 6-35(b)。其中:

(1) $\overline{\text{INC}}$ 是增加(或减小)输入脚,下降沿触发。触发 $\overline{\text{INC}}$ 将使滑动端向计数器增加或减少的方向移动,移动的方向由 $\text{U}/\overline{\text{D}}$ 端输入的逻辑电平决定。

(2) $\text{U}/\overline{\text{D}}$ 是升/降输入脚。 $\text{U}/\overline{\text{D}}$ 输入脚控制被调整的电容值的方向(即控制计数器是增加或减少)。

(3) Vcc 、GND 是电源和地。

(4) C_p 是 X90100 可调电容的一个固定端(等效于机械可调整电容器的固定端)。其最小直流电压是 0 V, 而最大电压是 V_{cc} 。两个端点的电容值由数字输入端 \overline{INC} 、 U/D 和 \overline{CS} 决定。

(5) C_m 是 X90100 可调电容的一个固定端(等效于机械可调整电容器的固定端)。

(6) \overline{CS} 是片选端, 低电平有效。当 \overline{CS} 变为高电平, 且 \overline{INC} 输入端也为高电平时, 当前计数器的值被储存于非易失性存储器中。当储存操作完成后 X90100 将处于低功耗的等待方式直到器件再次被选中。

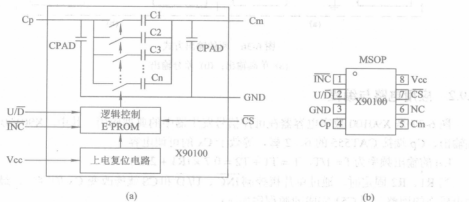


图 6-35 X90100 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

3. 器件的操作

\overline{INC} 、 U/D 和 \overline{CS} 三个输入信号控制着电容器总值的变化。只有 \overline{CS} 置低, 器件被选中才能响应 U/D 和 \overline{INC} 输入端的信号。在 \overline{INC} 输入端由高至低的变化将增加或减少(这取决于 U/D 输入端的状态)一个 5 位计数器的值。这个计数器的输出被译码并选择 32 个电容中的一个。只要 \overline{CS} 转变为高电平, 而这时 \overline{INC} 输入端也为高电平, 计数器的值就被储存在非易失性存储器中。X90100 的操作模式见表 6-9。

表 6-9 X90100 的操作模式

\overline{CS} 状态	\overline{INC} 状态	U/D 状态	模 式
低电平	由高向低(下降沿)	高电平	电容值增加一个增量
低电平	由高向低(下降沿)	低电平	电容值减小一个增量
由低向高(上升沿)	高电平	低(高)电平	存储当前位置
由低向高(上升沿)	低电平	高电平	不存储, 返回待机状态
高电平	高电平	高电平	进入待机状态

在 \overline{CS} 保持为低时, U/D 的状态可以改变。这种操作允许随时增加或减小计数器的内容, 即任意改变电容值, 直到得到合适的电容值为止。

4. 电容值的输出方式

X90100 可调电容有单端和差分两种输出方式。其连接实例如图 6-36 所示。

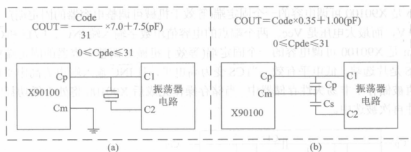


图 6-36 两种输出方式

(a) 单端输出; (b) 差分输出

6.9.2 应用电路与编程

图 6-37 是 X90100 数字电容器在可控信号发生器中的典型应用。其中, X90100 采用单端输出, C_p 端接 CAT555 的 6、2 脚, 等效于 C_x 的可调电容。

U_o 的输出频率为 $f = 1/T$, $T = T_1 + T_2 = 0.7 \times (R_1 + 2R_2) \times C_x$ 。

当 R_1 、 R_2 固定时, 通过单片机控制 \overline{INC} 、 U/\overline{D} 和 \overline{CS} 就能改变 C_x 的容量, 最后实现输出频率的调整。用 C51 完成的源程序如下:

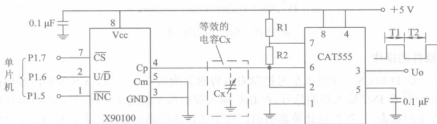


图 6-37 X90100 与 CAT555 构成的可控振荡器

```
#include <reg51.h>

sbit INC=P1^5;          /*定义器件的增加/减小引脚*/
sbit U_D=P1^6;          /*定义器件的升/降引脚*/
sbit CS=P1^7;           /*定义器件的片选引脚*/

void delay(unsigned int k) /*延时函数*/
{
    unsigned int i;
    for (i=0; i<k; i++);
}

void X90100Init(void) /*初始化*/
{
    CS=1;              /*片选置高*/
    U_D=1; INC=1;      /*默认增加*/
}
```

```
/*电容增加一个微量函数*/
```

```
void INC_Cx(void)
```

```
{
    X90100Init(); /*置输入口为高电平*/
    U_D=1; CS=0; delay(5); /*增大电容一个增量*/
    INC=0; delay(5); /*INC产生下降沿*/
    CS=1; delay(5); /*不存储*/
    INC=1;
}
```

```
/*电容减小一个微量函数*/
```

```
void DEC_Cx(void)
```

```
{
    X90100Init(); /*置输入口为高电平*/
    U_D=0; CS=0; delay(5); /*向减少方向移动一步*/
    INC=0; delay(5); /*INC产生下降沿*/
    CS=1; delay(5); /*不存储*/
    INC=1;
}
```

```
void main(void) /*测试函数*/
```

```
{
    unsigned char i;
    X90100Init(); /*初始状态*/
    for(i=0; i<31; i++) /*在开机后电容跳到最小位置*/
        DEC_Cx(); /*向小方向调整*/
    INC_Cx(); /*电容增加一档，输出一种频率*/
    while(1);
}
```

第7章 其他集成电路的使用与编程

7.1 CS5460A 高性能单片电力电能器件

7.1.1 硬件与功能描述

CS5460A 是高度集成的模拟/数字转换器(ADC), 它将两个通道的 $\Delta\Sigma$ ADC、高速能量计算功能和—个串行接口集成在—个单芯片上, 能够用于精确测量和计算单相 2 线或 3 线功率表的能量、瞬时功率、Irms 和 Vrms。CS5460A 可与低成本的分流器或变压器相连来测量电流, 或与电阻分压器或变压器相连来测量电压。CS5460A 可与微处理器串行通信, 并具有与能量成比例的固定带宽的频率输出。加电后, 芯片开放所有功能, 包括在用户程序控制下的系统校准等。

1. 主要性能特点

- (1) 在 1000:1 的动态范围内读取精度为 $\pm 10\%$;
- (2) 能测量能量、功率、电流有效值、电压有效值和能量/脉冲输出;
- (3) 有相位补偿和系统校准功能;
- (4) 输入端适合接分流器或电路传感器;
- (5) 采用单 +5 V 或 ± 2.5 V 供电, 精度为 $\pm 10\%$;
- (6) 内部参考电压为 2.5 V, 最大为 $60 \times 10^{-6}/^{\circ}\text{C}$;
- (7) 内带电源监视器;
- (8) 内带看门狗定时器;
- (9) 采用 3 线串行接口。

2. 内部结构与引脚说明

CS5460A 是带有能量计算引擎的 CMOS 单芯片功率测量器件。CS5460A 内部包括—个可编程增益放大器、两个 $\Delta\Sigma$ 调制器、两个高速数字滤波器、两个高通滤波器、参考电源、串行接口、系统校准、能量计算、Irms、Vrms 和瞬态功率计算等电路, 其内部结构如图 7-1(a) 所示。

CS5460A 可以与分流器或电流互感器相连来测量电流, 也可以与电阻分压器或变压器相连来测量电压。为了适应不同分流器输入的要求, 电流通道设计有可编程增益放大器(PGA, $\times 10$ 和 $\times 50$), 它可使用户输入测量 150 mV 的有效值或 30 mV 的有效值信号。CS5460A 包括两个高速数字滤波器, 它以(MCLK/K)/1024 的速率(OWR)输出数据。每个通道都有—个高通滤波器, 它可以在能量计算前将输出信号的直流部分过滤掉。

续表

名 称	脚 号	管 脚 含 义
IIN-	15	差分电流负输入端
IIN+	16	差分电流正输入端
PFMON	17	模拟电源掉电检测端
NC	18	空脚
RESET	19	器件复位端, 低电平有效
INT	20	当INT为低电平时, 表明内部寄存器数值已更新, 可以通过命令清除INT
EOUT	21	能量脉冲输出
EDIR	22	能量方向指示输出
SDI	23	串行接口数据输入端
XIN	24	晶振输入端, 往往与1脚配合连接晶振

3. 串口的操作

1) 操作的命令

CS5460A 的串口部分集成了一个待发送/接收的状态机, 状态机在 SCLK 的上升沿解释 8 位命令字。除了对命令解码外, 还为地址寄存器的数据传输做准备。写数据操作要在 8 位命令后, 继续写入 24 位数据(需 24 个时钟), 传送命令或数据时高位在前, 写时序如图 7-2(a)所示, 读时序如图 7-2(b)所示。所有的命令字长度都为 1 个字节, 数据长度为 3 个字节。

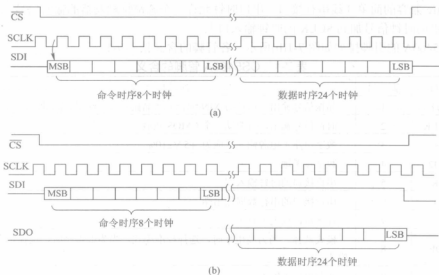


图 7-2 CS5460A 读/写时序

(a) SDI 写时序; (b) SDO 读时序

CS5460A 操作的几种命令格式如下:

(1) 启动命令的二进制格式为: (1、1、1、0、C、0、0、0)。其中, C 为 0 时执行计算单周期, C 为 1 时执行连续计算周期。

(2) SYNC0 命令的二进制格式为: (1、1、1、1、1、1、0)。这个命令是串口重新初始化序列的结束部分, 它也可以作为 NOP 命令, 串口可以通过发出 3 个以上 SYNC1 加一个 SYNC0 来使串口与字节界限重新同步。

(3) SYNC1 命令的二进制格式为: (1、1、1、1、1、1、1)。这个命令是串口重新初始化序列的一部分, 它也可以作为 NOP 命令, 但不能传输 3 个以上的连续字节。

(4) 加电/停止控制命令的二进制格式为: (1、0、1、0、0、0、0、0)。如果装置掉电, 则这个命令将使设备加电。在通电时, 不进行计算。如果已通电, 则所有计算被终止。

(5) 掉电控制命令的二进制格式为: (1、0、0、S1、S0、0、0、0)。其中, S1、S0 表示掉电模式, 当 S1、S0 为 01 时停止并进入待命功耗节省模式; 当 S1、S0 为 10 时, 停止并进入睡眠功耗节省模式。

(6) 校准控制命令的二进制格式为: (1、1、0、Cv、Ci、0、GC、OC)。其中, Cv、Ci 用来指定校准通道, 当 Cv、Ci 为 01 时, 为校准电流通道; 当 Cv、Ci 为 10 时, 为校准电压通道; 当 Cv、Ci 为 11 时, 为电压通道、电流通道同时校准。GC 位用来指定增益校准, GC 为 0 时正常运行; GC 为 1 时执行增益校准。OC 位用来指定偏置校准, 当 OC 为 0 时, 正常运行; 当 OC 为 1 时, 偏置校准。

(7) 寄存器读/写命令位的二进制格式为: (0、写/读、RA4、RA3、RA2、RA1、RA0 和 0)。其中, 写/读为 0 时, 读取寄存器操作, 写/读为 1 时, 写寄存器操作; RA4~RA0 是寄存器地址位。表 7-2 列出了 CS5460A 寄存器的地址与名称。

表 7-2 CS5460A 寄存器的地址与名称

地址	名称	含义	地址	名称	含义
00000	Config	配置寄存器	00001	Loff	电流偏置校准
00010	Lgn	电流增益校准	00011	Voff	电压偏置校准
00100	Vgn	电压增益校准	00101	CycleCount	转换成整数的数目(N)
00110	Pulse-Rate	校准频率输出	00111	I	上一次电流值
01000	V	上一次电压值	01001	P	上一次功率值
01010	E	上一次能量值	01011	Irms	上一个电流有效值
01100	Vrms	上一个电压有效值	01101	TBC	时基校准
01110	Tset	只能内部使用	01111	Status	状态寄存器
10000	Res	转换	10001	Res	转换
11000	Test	只能内部使用	11001	Test	只能内部使用
11010	Mask	中断屏蔽寄存器	11011	Test	只能内部使用
11100	Res	转换	11101	Res	转换
11110	Res	转换	11111	Res	转换

2) 接口信号

CS5460A 串口包括 4 个控制线: \overline{CS} 、SDI、SDO 和 SCLK。其中, \overline{CS} 为片选信号, 低电平有效; SDI 是串行数据输入信号; SDO 是串行数据输出信号。当 \overline{CS} 为逻辑 1 时, SDO 输出为高阻。SCLK 是串行时钟, 在 CS5460A 的内部设计有施密特触发器对时钟信号进行整形。串口读/写时序如图 7-2 所示。

3) 串口初始化

无论是执行复位还是端口初始化, 串口都会被设为命令模式。端口初始化序列包括 3 个(或更多)AYNC1 命令字节(0xFF), 后面是 SYNC0 命令字节(0xFE)。这个序列使芯片进入命令模式, 并等待有效的命令。

4) 系统初始化

可以在任意时刻进行一个软件或硬件的复位。软件复位是通过向配置寄存器的 RS 位写入逻辑 1 来实现的, 复位以后自动恢复为逻辑 0。在第 32 个 SCLK 结束时, 内部按 3 或 4 个 DCLK(MCLK/K)来同步延迟配置寄存器。然后复位电路在 MCLK 的第一个下降沿开始复位。硬件复位是通过强制给 RESET 管脚加上最小 50 ns 宽的低电平来实现的。RESET 信号是异步的, 不需要 MCLK 同步。

系统复位后, 配置寄存器为 0x000001, 偏置寄存器为 0x000000, 增益寄存器为 0x400000, 脉冲速率寄存器为 0x0FA000, 周期计数寄存器为 0x000FA0, 时基寄存器为 0x800000, 状态寄存器为 0x000001, 屏蔽寄存器为 0x000000, 符号寄存器为 0x000000, 无符号寄存器为 0x000000。

4. 寄存器描述

1) 配置寄存器

配置寄存器的 24 位格式如表 7-3 所示。

表 7-3 配置寄存器的 24 位格式(地址为 0x0, 缺省值是 0x000001)

位 23	位 22	位 21	位 20	位 19	位 18	位 17	位 16
PC6	PC5	PC4	PC3	PC2	PC1	PC0	Gi
位 15	位 14	位 13	位 12	位 11	位 10	位 9	位 8
EWA	Res	Res	SI1	SI0	EOD	DL1	DL0
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
RS	VHPF	IHPF	iCPU	K3	K2	K1	K0

对表 7-3 的说明如下:

(1) K[3..0]是内部时钟分配系数。内部时钟频率 $DCLK=MCLK/K$, K 的取值可以是 1~16。当 K 为 0 时, 默认为 16。

(2) iCPU 位是 CPUCLK 时钟噪声控制位。为了减少模拟信号取样的噪声, 在取样沿 CPUCLK 应无效。当 iCPU 位设为 0 时, 为正常模式; 当 iCPU 位设为 1 时, 在 CPUCLK 上升沿使噪声减小。

(3) IHPF 位是电流通道高通滤波器控制位。当 IHPF 位设为 0 时, 电流通道高通滤波器被禁止; 当 IHPF 位设为 1 时, 高通滤波器有效。

(4) VHPF 位是电压通道高通滤波器控制位。当 VHPF 位设为 0 时, 电压通道高通滤波器被禁止; 当 VHPF 位设为 1 时, 高通滤波器有效。

(5) RS 位是芯片的复位控制位。当 RS = 1 时, 启动一次芯片复位, 复位完成后自动将 RS 清零。

(6) DL0 位是 EDIR 引脚定义位。当 EOD = 1 时, 用 DL0 位设置 EDIR 引脚电平, DL0

的缺省值为 0。

(7) DL1 位是 $\overline{\text{EOUT}}$ 引脚定义位。当 $\text{EOD} = 1$ 时, 用 DL0 位设置 $\overline{\text{EOUT}}$ 引脚电平, DL1 的缺省值为 0。

(8) EOD 位是允许 DL0 和 DL1 控制 $\overline{\text{EDIR}}$ 、 $\overline{\text{EOUT}}$ 管脚位。 $\overline{\text{EDIR}}$ 、 $\overline{\text{EOUT}}$ 管脚可以通过状态寄存器访问。当 $\text{EOD} = 0$ 时, $\overline{\text{EDIR}}$ 、 $\overline{\text{EOUT}}$ 管脚正常输出; 当 $\text{EOD} = 1$ 时, $\overline{\text{EDIR}}$ 、 $\overline{\text{EOUT}}$ 管脚由 DL0 与 DL1 位控制。

(9) SI[1..0] 位是软中断设置位。该位用来选择适当方式作为中断。当 SI[1..0] = 00 时, 激活低电平中断(缺省); 当 SI[1..0] = 01 时, 激活高电平中断; 当 SI[1..0] = 10 时, 下降沿中断(INT 正常处于高电平); 当 SI[1..0] = 11 时, 上升沿中断(INT 正常处于低电平)。

(10) Res 是保留位。Res 位必须设为 0。

(11) EWA 是允许多个 $\overline{\text{EDIR}}$ 和 $\overline{\text{EOUT}}$ 管脚在一起“线与”位。 $\text{EWA} = 0$ 时, 为正常输出; $\text{EWA} = 1$ 时, 激活 $\overline{\text{EDIR}}$ 、 $\overline{\text{EOUT}}$ 管脚推挽状态。

(12) Gi 是电流 PGA 增益位。Gi = 0 时电流增益是 10(缺省); Gi = 1 时电流增益是 50。

(13) PC[6..0] 是相位补偿位。该位用来补偿电压通道的延时, 该位的数值大小与电压通道延时成正比。当 $\text{MCLK/K} = 4.096 \text{ MHz}$, $K = 1$ 时, 相位调整范围是 $-2.8^\circ \sim +2.8^\circ$, 每步 0.04° (这正好是 60 Hz 的线性度), PC[6..0] 的缺省值是 0 (在 $\text{MCLK/K} = 4.096 \text{ MHz}$ 时, 为 0.0125° 延时)。

2) 电流通道和电压通道直流偏置寄存器

电流通道和电压通道偏置寄存器的 24 位格式如表 7-4 所示。

表 7-4 电流通道(地址 0x1)和电压通道(地址 0x3)偏置寄存器的 24 位格式, 缺省值是 0.000

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

复位时, 偏置寄存器的值为 0, 允许测量。所存的值可以进行适当的补偿。

3) 电流通道增益和电压通道增益寄存器

电流通道增益和电压通道增益寄存器的 24 位格式如表 7-5 所示。

表 7-5 电流通道(地址 0x2)和电压通道(地址 0x4)增益寄存器的 24 位格式, 缺省值是 1.000

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^1	2^0	2^{-1}	2^{-2}	...	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}

复位时, 偏置寄存器的值为 1.000, 允许测量。所存的值可以进行适当的补偿。

4) 周期计数寄存器

周期计数寄存器的 24 位格式如表 7-6 所示。

表 7-6 周期计数寄存器(地址 0x5)的 24 位格式, 缺省值是 4000

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^{23}	2^{22}	2^{21}	2^{20}	...	2^4	2^3	2^2	2^1	2^0

复位时, 周期计数器的默认值为 4000。

周期计数器的值决定能量和有效值的转换长度。转换周期的计算公式是 $(\text{MCLK/K})/(1024 \times N)$, 其中, MCLK 是主时钟, N 是周期数。为了计算电压有效值、电流有

效值和能量, N 必须大于 10。

5) 脉冲速率寄存器

脉冲速率寄存器的 24 位格式如表 7-7 所示。

表 7-7 脉冲速率寄存器(地址 0x6)的 24 位格式, 缺省值是 32000.00 Hz

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^{18}	2^{17}	2^{16}	2^{15}	...	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}

复位时, 脉冲速率寄存器默认值为 32000.00 Hz。

脉冲寄存器的值决定了 $\overline{\text{EOUT}}$ 引脚的脉冲输出速率。每个 $\overline{\text{EOUT}}$ 脉冲代表一个额定的能量。

6) 符号寄存器

I、V、P、E 符号输出寄存器的 24 位格式如表 7-8 所示。

表 7-8 I、V、P、E 符号输出寄存器(地址 0x7~0xA)的 24 位格式

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

符号寄存器包括 I、V、P、E 的上一次测量结果, 如果值域为 $-1.0 \leq I, V, P, E < 1.0$, 则使用二进制补码表示, 小数点在 MSB(符号位)右侧。I、V、P、E 是包括带符号值的输出结果寄存器。

7) 电流、电压无符号输出寄存器

电流、电压无符号输出寄存器的 24 位格式如表 7-9 所示。

表 7-9 电流、电压无符号输出寄存器(地址 0xB~0xC)的 24 位格式

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^{-1}	2^{-2}	2^{-3}	2^{-4}	...	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^{-24}

电流、电压无符号输出寄存器包括上一次测量结果, 如果值域为 $0.0 \leq \text{电流}, \text{电压} < 1.0$, 则使用二进制补码表示, 小数点在 MSB 左侧, 电流、电压是包括无符号值的输出寄存器。

8) 时基校准寄存器

时基校准寄存器的 24 位格式如表 7-10 所示。

表 7-10 时基校准寄存器(地址 0xD)的 24 位格式, 缺省值是 1.000

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

复位时, 时基校准寄存器的默认值为 1.000。

时基校准寄存器用以校准晶片振荡器所造成的误差, 值域为 $0.0 \leq \text{TBC} < 2.0$ 。

9) 功率偏置寄存器

功率偏置寄存器的 24 位格式如表 7-11 所示。

表 7-11 功率偏置寄存器(地址 0xE)的 24 位格式, 缺省值是 0.000

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

复位时, 偏置寄存器的值为 0, 允许测量。所存的值可以进行适当的补偿。

10) 电流通道和电压通道交流偏置寄存器

电流通道和电压通道交流偏置寄存器的 24 位格式如表 7-12 所示。

表 7-12 电流通道(地址 0x10)和电压通道(地址 0x11)交流偏置寄存器的 24 位格式, 缺省值是 0.000

位 23	位 22	位 21	位 20	...	位 4	位 3	位 2	位 1	位 0
2^{-13}	2^{-14}	2^{-15}	2^{-16}	...	2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}

复位时, 偏置寄存器的值为 0.000, 允许测量。所存的值可以进行适当的补偿。

11) 状态和屏蔽寄存器

状态和屏蔽寄存器的 24 位格式如表 7-13 所示。状态寄存器的缺省值是 0x000001, 屏蔽寄存器的缺省值是 0x000000。

表 7-13 状态(地址为 0xF)和屏蔽寄存器(地址 0x1A)的 24 位格式

位 23	位 22	位 21	位 20	位 19	位 18	位 17	位 16
DRDY	EOUT	EDIR	CRDY	MATH	Res	IOR	VOR
位 15	位 14	位 13	位 12	位 11	位 10	位 9	位 8
PWOR	IROR	VROR	EOR	EOOR	Res	ID3	ID2
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
ID1	ID0	WDT	VOD	IOD	LSD	0	IC

状态寄存器用来指示器件的状态, 通常情况下, 写一个“1”到某位, 会使该位变成“0”状态, 写一个“0”到某位, 会使该位内容不变。通过这种性能可以清除某位的内容。

屏蔽寄存器用来控制 INT 的状态。向屏蔽寄存器写一个“1”, 当状态位有效时, 将激活 INT 管脚。

对表 7-13 的说明如下:

(1) IC 位是无效命令位, 正常值为逻辑 1, 当命令无效时置为逻辑 0。只有在向端口发出一个端口初始序列时才无效。当写入状态寄存器时, IC 位被忽略。

(2) LSD 是低电源检测位, 当 PFMON 脚下降到相对于 VA-脚低于 2.5 V 时, LSD 位置“1”。

(3) IOD 是电流通道调制器振荡检测位, 当调制器输入高于满量程时, IOD 位置“1”。

(4) VOD 是电压通道调制器振荡检测位, 当调制器输入高于满量程时, VOD 位置“1”。

(5) WDT 是看门狗读取位, 能量寄存器超过 5 秒钟没有读取数据时, WDT 被置位 (MCLK = 4.096 MHz, K = 1)。为了清除 WDT 位, 首先要读取能量寄存器数据, 然后将 WDT 位设为逻辑“1”, 写入状态寄存器。

(6) ID[3..0]是版本或 ID 标识位。

(7) EOOR 是 EOUT 能量寄存器超出标志位, 原因可能是输出字速率太低, 以至于无法测量。可以通过设定一个脉冲速率寄存器较高的频率来解决这个问题。

(8) EOR 是能量超出范围位, 当校准能量值过大或过小时被置位。

(9) VROR 是电压有效值超出范围位, 当校准电压值过大时被置位。

(10) IROR 是电流有效值超出范围位, 当校准电流值过大时被置位。

- (11) PWOR 是功率计算超出范围位。
- (12) VOR 是电压超出范围位。
- (13) IOR 是电流超出范围位。
- (14) MATH 是计算出错位(例如被 0 除)。
- (15) CRDY 是转换读标志位。
- (16) EDIR 是能量和小于 0 时被置位(与 EOUT 同时置位或清除)。
- (17) EOUT 指能量与频率转换达到能量的界限时, $\overline{\text{EOUT}}$ 脚产生一个脉冲串。
- (18) DRDY 是数据就绪(准备好), 在校准或转换周期结束后被置位。

12) 控制寄存器

控制寄存器的 24 位格式如表 7-14 所示。

表 7-14 控制寄存器(地址为 0x1C)的 24 位格式, 缺省值为 0x000000

位 23	位 22	位 21	位 20	位 19	位 18	位 17	位 16
Res	Res	Res	Res	Res	Res	Res	Res
位 15	位 14	位 13	位 12	位 11	位 10	位 9	位 8
Res	Res	Res	Res	Res	Res	Res	STOP
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
Res	MECH	Res	INTL	SYNC	NOCPU	NOOSC	STEP

对表 7-14 的说明如下:

- (1) STOP 是停止 EEBOOT 位, 当 STOP 设为 1 时停止新的 EEBOOT 时序。
- (2) Res 是保留位, 必须设为 0。
- (3) MECH 是展宽输出频率宽度位, 当 MECH = 1 时, 展宽 $\overline{\text{EOUT}}$ 和 $\overline{\text{EDIR}}$ 输出脉冲宽度。
- (4) INTL 是设置 $\overline{\text{INT}}$ 管脚输出方式位, 当 INTL = 1 时, $\overline{\text{INT}}$ 管脚输出设为开路方式。
- (5) SYNC 是内部 A/D 转换同步位。当 SYNC = 1 时, 迫使 A/D 转换与初始化命令同步。
- (6) NOCPU 是 CPUCLK 输出控制位。当 NOCPU = 1 时, 可减少功耗。
- (7) NOOSC 是控制外部晶振位。NOOSC = 1 时取消外部晶振驱动。
- (8) STEP 是步进控制位。STEP = 1 时, 允许 $\overline{\text{EOUT}}$ 和 $\overline{\text{EDIR}}$ 脉冲输出。

5. 部分功能描述

1) 参考电压

CS5460A 的参考电压可以由两种方式获取。

- (1) 将 VREFIN 和 VREFOUT 相连, 使用内部基准。内部基准电压精度不高($60 \times 10^{-6} / ^\circ\text{C}$)。
- (2) 使用外部基准。当内部基准精度不够时, 可用外部基准(如 LT1019-2.5 V), 基准从 VREFIN 输入。

2) 校准

CS5460A 提供了系统偏置和系统增益两个直流校准模式。对于系统校准, 用户必须向转换器提供表示地和满量程的校准信号。用户在执行系统增益校准时, 必须提供正极大满量程信号。一个校准周期结束后, DRDY 被置位, 校准结果被存放在增益寄存器和偏置寄存器中。

图 7-3(a)为偏置系统校准示意图。图 7-3(b)为增益系统校准示意图。注意, 为了使设备周围的数字噪声最小, 用户在读/写串行端口时应等待每个校准过程完成, 且偏置校准应该在增益校准前完成。

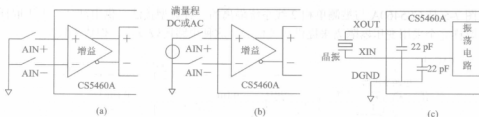


图 7-3 校准与晶振电路

(a) 偏置校准; (b) 增益校准; (c) 晶振电路

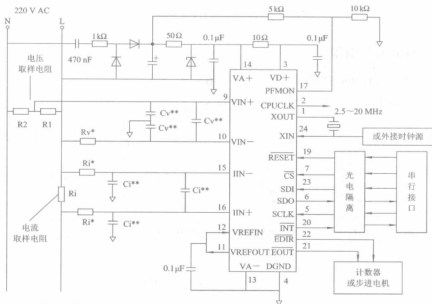
3) 振荡器的特性

CS5460A 的 XIN 和 XOUT 分别是一个反相放大器的输入与输出,通常外接一个石英晶振或陶瓷振荡器工作,如图 7-3(c)所示。当使用外部时钟时,信号从 XIN 脚输入, XOUT 悬空。

CS5460A 可由 3~20 MHz 时钟驱动。在设置分割系数 K(缺省值为 1)的取值后,内部时钟的范围为 2.3~10 MHz。为了正常工作,时钟必须适当编程。

7.1.2 典型接法

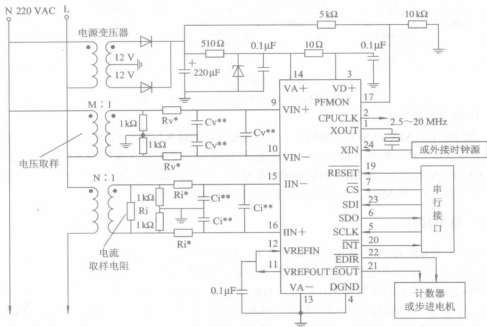
图 7-4 是 CS5460A 在单电源模式下测量单相 2 线系统功率的接法。其中, 串行接口通过光电耦合与微机系统连接; 电压的取样通过 R1、R2 的分压得到; 电压差分输入端 VIN+ 和 VIN- 的电容是根据实际应用环境确定的滤波电容; Ri 是电流取样电阻, 与其有关的电阻、电容也是根据实际情况确定的滤波网络。基准采用内部提供的 +2.5 V 电压, 如果对精度要求更高, 则可实用外接高精度基准电源。系统时钟既可由晶振提供, 也可外接时钟源从 24 脚输入。如果直接计数或驱动步进电机, 则信号从能量输出 EOUT 信号获得。



注: **表示滤波电容,*表示输入保护。

图 7-4 单相 2 线连接图

图 7-5 是 CS5460A 与被测单相 2 线系统隔离的一种典型接法。供电部分、电压取样、电流取样完全采用变压器隔离来提供相关信号。这种电路比较安全，但成本较高。



注：**表示滤波电容，*表示输入保护。

图 7-5 采用隔离的单相 2 线连接图

7.1.3 应用电路与编程

CS5460A 与 STC89C52 的接口电路如图 7-6 所示。

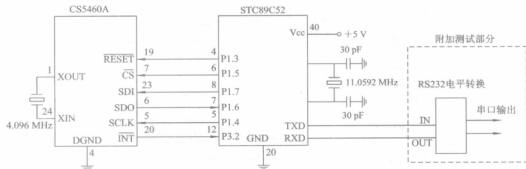


图 7-6 CS5460A 与 STC89C52 的接口电路

根据图 7-6 所示的 SPI 总线接口及 CS5460A 的工作原理，用 C51 编写的源程序如下：

```
#include <reg52.h>
```

```
/*包含 STC89C52 的头文件*/
```

```

#include <stdio.h> /*包含输入/输出头文件*/

sbit CS5460_nCS = P1^5; /*定义片选引脚*/

sbit CS5460_SCLK=P1^4; /*定义 SPI 总线时钟引脚*/

sbit CS5460_SDI =P1^7; /*定义 SPI 总线输入引脚*/

sbit CS5460_SDO =P1^6; /*定义 SPI 总线输出引脚*/

sbit CS5460_nRST=P1^3; /*定义 CS5460A 复位引脚*/

/*根据 CS5460A 的工作原理的常用配置如下*/

#define CS5460_CFG_K 0x01L /*1~16, 0 代表 16*/

#define CS5460_CFG_IHPF 0x0L<<5 /*0:disable, 1:enable, 电流高通滤波器(第 5 位)*/

#define CS5460_CFG_VHPF 0x0L<<6 /*0:disable, 1:enable, 电压高通滤波器(第 6 位)*/

#define CS5460_CFG_GI 0x0L<<16 /*0:X10, 1:X50, 电流增益(第 16 位)*/

#define CS5460_CFG_OTHER 0x001410L //INT:下降沿; 不输出脉冲; (Refer to datasheet)

/*将以上配置汇总到 Config 寄存器*/

#define CS5460_CFG CS5460_CFG_K | CS5460_CFG_IHPF | CS5460_CFG_VHPF | \
CS5460_CFG_GI | CS5460_CFG_OTHER /*控制寄存器*/

#define CS5460_CTRL 0x000014

/*累加计数器*/

#define CS5460_CYC_CNT 0x001000

/*MCLK 频率, 单位为 Hz*/

#define MCLK_FERQ 4096000

/*RMS 和 Energy 计算频率 COMP_FREQ, 单位为 Hz*/

#if CS5460_CFG_K!=0

#define COMP_FREQ (MCLK/CS5460_CFG_K)/(CS5460_CYC_CNT*1024)

#else

#define COMP_FREQ (MCLK/16)/(CS5460_CYC_CNT*1024)

#endif

/*电流取样变换系数=输入电流/采样电压.浮点型, 单位为 A/V*/

#define I_TRANS_CO 0.00212766

/*电压取样变换系数=输入电压/采样电压.浮点型, 单位为 V/V*/

#define V_TRANS_CO 2.0

/*寄存器地址, 与“读/写”命令地址位右对齐*/

#define CS5460_ADD_CFG 0x00<<1 /*配置寄存器地址*/

#define CS5460_ADD_IDC_OFF 0x01<<1 /*电流偏置校准*/

#define CS5460_ADD_I_GAIN 0x02<<1 /*电流增益校准*/

#define CS5460_ADD_VDC_OFF 0x03<<1 /*电压偏置校准*/

#define CS5460_ADD_V_GAIN 0x04<<1 /*电压增益校准*/

#define CS5460_ADD_CYC_CNT 0x05<<1 /*A/D 转换数*/

#define CS5460_ADD_PUL_RATE 0x06<<1 /*频率输出*/

#define CS5460_ADD_I 0x07<<1 /*电流值*/

```



```

#define CS5460_ADD_V      0x08<<1 /*电压值*/
#define CS5460_ADD_P      0x09<<1 /*功率值*/
#define CS5460_ADD_E      0x0a<<1 /*能量值*/
#define CS5460_ADD_I_RMS  0x0b<<1 /*电流有效值*/
#define CS5460_ADD_V_RMS  0x0c<<1 /*电压有效值*/
#define CS5460_ADD_TBC    0x0d<<1 /*校准值*/
#define CS5460_ADD_P_OFF  0x0e<<1 /*功率偏差*/
#define CS5460_ADD_STATUS 0x0f<<1 /*状态值*/
#define CS5460_ADD_IAC_OFF 0x10<<1 /*电流 AC 偏差*/
#define CS5460_ADD_VAC_OFF 0x11<<1 /*电压 AC 偏差*/
#define CS5460_ADD_CON     0x12<<1 /*转换*/
#define CS5460_ADD_MASK    0x1a<<1 /*标志*/
#define CS5460_ADD_CTRL    0x1c<<1 /*控制*/

/*命令*/
#define CS5460_CMD_SIN_CONV 0xe0 /*单周期*/
#define CS5460_CMD_CON_CONV 0xe8 /*连续周期*/
#define CS5460_CMD_SYNC0    0xfe
#define CS5460_CMD_SYNC1    0xff
#define CS5460_CMD_POW_UP    0xa0
#define CS5460_CMD_STA_BY    0x88 /*允许快速加电*/
#define CS5460_CMD_POW_SAVE 0x90 /*停止进入睡眠*/

/*同时校准直流 V, I 增益, 必须根据应用改变校准类型*/
#define CS5460_CMD_CAL_GAIN 0xda
/*同时校准直流 V, I 偏置, 必须根据应用改变校准类型*/
#define CS5460_CMD_CAL_OFF  0xd9
/*读寄存器命令掩码, 跟地址相“或”后得到完整命令*/
#define CS5460_CMD_READ     0x00
/*写寄存器命令掩码, 跟地址相“或”后得到完整命令*/
#define CS5460_CMD_WRITE    0x40
/*名称: CS5460A_Write*/
/*功能: 向 CS5460A-BS 写入数据*/
/*参数: cmd 为写入命令, 不包括不带参数的命令。不带参数的命令*/
/*使用 CS5460A_command()函数写入。data 为带写入的数据, 长整数, 低 3 字节有效。*/
void CS5460A_Write(unsigned char cmd,unsigned long int dat)
{
    unsigned char i;
    dat<=8; /*将数据低 3 字节向左平移至高 3 字节*/
    CS5460_SDO=1; /*SDO 引脚设为输入状态*/
    CS5460_nCS=0; /*芯片使能*/

```

```

/*写入命令(8位)*/
for(i=0; i<8; i++)
{
    CS5460_SDI=(bit)(cmd & 0x80);          /*准备数据位*/
    CS5460_SCLK=0;
    cmd<<=1;
    CS5460_SCLK=1;          /*时钟上升沿写入数据*/
}
for(i=0; i<24; i++)          /*写入数据(24位)*/
{
    CS5460_SDI=(bit)(dat & 0x80000000);    /*准备数据位*/
    CS5460_SCLK=0;
    dat<<=1;
    CS5460_SCLK=1;          /*时钟上升沿写入数据*/
}
CS5460_SDI=1;
CS5460_SCLK=0;
CS5460_nCS=1;
CS5460_SCLK=1;
}

/*名称: CS5460A_Read*/
/*功能: 从 CS5460A-BS 读出数据*/
/*参数: addr 为寄存器地址*/
/*return 为带读出的数据, 长整数, 低 3 字节有效。*/
unsigned long int CS5460A_Read(unsigned char addr)
{
    unsigned char i;
    unsigned long int retn=0;
    unsigned long int syncmd=0xfefefeff;    /*3 个 SYNC0 命令和 1 个 0xff*/
    addr=CS5460_CMD_READ;
    CS5460_SDO=1;          /*SDO 引脚设为输入状态*/
    CS5460_SCLK=0;
    CS5460_nCS=0;          /*芯片使能*/
    /*写入命令(8位)*/
    for(i=0; i<8; i++)
    {
        CS5460_SCLK=0;
        CS5460_SDI=(bit)(addr & 0x80);    /*准备数据位, RLC A*/
        addr<<=1;

```

```

        CS5460_SCLK=1;           /*时钟上升沿写入数据*/
    }
    CS5460_SCLK=0;
    for(i=0; i<24; i++)           /*读出数据*/
    {
        CS5460_SDI=(bit)(syncmd & 0x80000000);
        syncmd<<=1;
        CS5460_SCLK=1;
        retn<<=1;
        if(CS5460_SDO) retn|=1;
        CS5460_SCLK=0;
    }
    CS5460_nCS=1;
    CS5460_SDI=1;
    CS5460_SCLK=1;
    return(retn);
}

```

/*名称: CS5460A_TypeConv*/

/*功能: 将 ADC 编码转换成实际物理量的值*/

/*参数: type 为 dat 的类型。类型名与 CS5460 中对应地址名一致, 比如电压的类型*/

/*CS5460_ADD_V dat 为 CS5460A 转换结果, 无符号整型, 低 3 字节有效*/

/*返回值: dat 对应实际物理量的值, 浮点型, 单位为 mV/mA/mW/mJ*/

float CS5460A_TypeConv(unsigned char type,unsigned long int dat)

```

{
    long int temp;
    float result;
    temp=(long int)(dat<<8); /*24bit signed long -> 32bit signed long 类型转换*/
    temp=temp/256;
    switch(type)             /*单位换算*/
    {
        case CS5460_ADD_I_RMS :
            temp=(long int)dat; /*RMS 为无符号数, 重新转换格式*/
        case CS5460_ADD_I :
            if((CS5460_CFG_GI>>16)==0)
                result=I_TRANS_CO*(250.0*temp)/8388607;
            else
                result=I_TRANS_CO*(5.0*temp)/8388607;
            break;
        case CS5460_ADD_E :

```

```

case CS5460_ADD_P:
    result=250*(250.0*temp)*V_TRANS_CO*I_TRANS_CO/8388607;
    break;
case CS5460_ADD_V_RMS:
    temp=(long int)dat;          /*RMS 为无符号数，重新转换格式*/
case CS5460_ADD_V:
    result=temp*(250.0*V_TRANS_CO/8388607);
    break;
default:
    result=33554432;
}
return(result);
}

void InitComm(void)          /*串口初始化*/
{
    SCON  = 0x50;
    TMOD |= 0x20;
    PCON |= 0x80;
    TH1   = 0xff;           /*TH1: reload value for 57600 b/s @ 11.0592 MHz*/
    TL1   = 0xff;
    TR1   = 1;
}

void Delayms(int ms)        /*延时函数*/
{
    int i,j;
    for(i=0; i<ms; i++)
        for(j=0; j<240; j++);
}

/*名称: CS5460A_Init*/
/*功能: 初始化 CS5460A, 参考“常用配置”*/
void CS5460A_Init(void)
{
    /*Step 1: 芯片复位(这里 Reset 引脚复位, 也可以使用软件复位)*/
    CS5460_nRST=0;
    Delayms(20);
    CS5460_nRST=1;

    /*Step 2: 写配置寄存器和控制寄存器, 相关参数在“常用配置”宏定义中更改*/
    CS5460A_Write(CS5460_CMD_WRITE|CS5460_ADD_CFG, CS5460_CFG);
    CS5460A_Write(CS5460_CMD_WRITE|CS5460_ADD_CTRL, CS5460_CTRL);
}

```

```

CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_CYC_CNT, CS5460_CYC_CNT);
/*Step 3: 写入校准参数(这里使用默认参数, 需根据实际情况保存并配置校准参数)*/
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_I_GAIN, 0x400000);
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_IDC_OFF, 0x000000);
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_V_GAIN, 0x400000);
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_VDC_OFF, 0x000000);
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_IAC_OFF, 0x000000);
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_VAC_OFF, 0x000000);
CS5460A_Write(CS5460_CMD_WRITE | CS5460_ADD_P_OFF, 0x000000);
/*Step 4: 启动连续转换*/
CS5460A_Write(CS5460_CMD_CON_CONV, 0xfffff);
}

/*测试部分*/
void main(void) /*主函数*/
{
    unsigned long int temp=0;
    float voltage,power,current,Vrms,Irms,Energy;
    InitComm(); /*串口初始化*/
    CS5460A_Init(); /*CS5460A 初始化*/
    /*读/写测试*/
    temp=CS5460A_Read(CS5460_ADD_V_GAIN);
    printf("Original %ld \n",temp);
    for(temp=120000; temp<898988; temp=temp+40000)
    {
        printf("Write %ld \n",temp);
        CS5460A_Write((CS5460_CMD_WRITE | CS5460_ADD_V_GAIN),temp);
        temp=0;
        temp=CS5460A_Read(CS5460_ADD_V_GAIN);
        printf("Read back %ld \n",temp);
    }
    while(1)
    {
        temp=CS5460A_Read(CS5460_ADD_V);
        voltage=CS5460A_TypeConv(CS5460_ADD_V,temp); /*得到电压*/
        temp=CS5460A_Read(CS5460_ADD_P);
        power=CS5460A_TypeConv(CS5460_ADD_P,temp); /*得到功率*/
        temp=CS5460A_Read(CS5460_ADD_I);
        current=CS5460A_TypeConv(CS5460_ADD_I,temp); /*得到电流*/
        temp=CS5460A_Read(CS5460_ADD_E);
    }
}

```

```

Energy=CS5460A_TypeConv(CS5460_ADD_E,temp);    /*得到能量*/
temp=CS5460A_Read(CS5460_ADD_I_RMS);
Irms=CS5460A_TypeConv(CS5460_ADD_I_RMS,temp);    /*得到电流有效值*/
temp=CS5460A_Read(CS5460_ADD_V_RMS);
Vrms=CS5460A_TypeConv(CS5460_ADD_V_RMS,temp);    /*得到电压有效值*/
Delays(500);
printf("Voltage %f\n",voltage);                    /*测试输出电压*/
printf("Current %f\n",current);                    /*测试输出电流*/
printf("Power %f/%f\n",power,voltage*current);    /*测试输出功率*/
printf("Vrms %f\n",Vrms);                        /*测试输出电压有效值*/
printf("Irms %f\n",Irms);                        /*测试输出电流有效值*/
printf("Energy per second %f\n",Energy);          /*输出能量*/
printf("-----\n");
}
}

```

7.2 MAX7219 单片数码管驱动器

7.2.1 硬件与功能描述

MAX7219 是串行输入共阴极管显示驱动器,这种接口 CPU 可驱动 8 位 7 段数字型的 LED、条形图显示器或 64 只独立的 LED。MAX7219 内置一个 BCD 码译码器、多路扫描电路、段和数字驱动器及一个存储每一位的 8×8 静态 RAM。对所有的 LED 来说,只需外接一个电阻即能控制段电流的大小。

MAX7219 内有一个 $150 \mu\text{A}$ 的低功耗掉电模式和多种数控电路,提供有显示位数(1~8 位)可选的界限扫描寄存器,允许用户为每一位选择 BCD 译码或不译码方式。

该器件可广泛应用于条形图显示、7 段显示、工业控制、仪表控制面板和 LED 模型显示等领域。

1. 主要性能特点

- (1) 串行接口频率: 典型值 10 MHz;
- (2) 低功耗模式: $<150 \mu\text{A}$ (保存数据);
- (3) 连续功耗: 窄 DIP 封装约 0.87 W, 宽 SO 封装约 0.76 W, 窄 Cerdip 封装约 1.1 W;
- (4) DIG0~DIG7 吸收电流: $<500 \text{ mA}$;
- (5) 工作电压: $+4.0 \sim +5.5 \text{ V}$;
- (6) 工作电流: 典型值 320 mA;
- (7) 显示扫描速度: 500~1300 Hz(8 位 LED 扫描);
- (8) 位驱动吸收电流: $>320 \text{ mA}$;
- (9) 工作温度: $0 \sim +70^\circ\text{C}$ (民品)、 $-45 \sim +85^\circ\text{C}$ (工业级)。

2. 内部结构与引脚说明

MAX7219 的内部组成如图 7-7 所示。

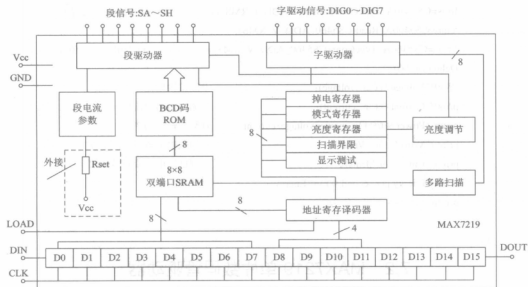


图 7-7 MAX7219 的内部组成

MAX7219 外形采用 24 脚 DIP 封装，其引脚排列如图 7-8(a)所示，引脚功能如表 7-15 所示。

表 7-15 MAX7219 的引脚功能说明

管脚名称	管脚	功能描述
DIN	1	串行数据输入，在 CLK 的上升沿数据被内部采样
CLK	13	串行时钟，上升沿有效
DIG0~DIG7	2, 3, 5~8, 10, 11	8 位数据数据“位”驱动输出，接数码管的 COM 端
GND	4, 9	信号地，4、9 脚必须连起来接“地”
LOAD	12	在 LOAD 的上升沿把最后 16 位数据装入锁存
SA~SH	14~17, 20~23	8 段数据输出，接数码管的段信号
ISSET	18	驱动电流调节，通过与电源串接一个电阻来调节最大段电流
Vcc	19	电源电压，接+5 V
DOUT	24	串行数据输出，用于级联，一般不用

MAX7219 的工作时序如图 7-8(b)所示，其串行数据格式见图 7-8(c)。

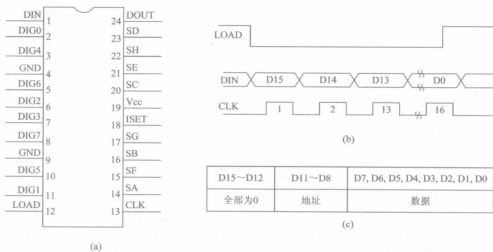


图 7-8 MAX7219 的引脚排列、工作时序和数据格式

(a) 引脚排列; (b) 工作时序; (c) 数据格式

7.2.2 MAX7219 的使用与设置

1. 串行寻址方式

对于 MAX7219, 在 LOAD 为低电平时, 将 16 位数据串发送到 DIN 端, 在每个 CLK 的上升沿把数据移入到内部 16 位寄存器中。DIN 端的数据通过移位寄存器传送, 并在 16.5 个时钟周期后出现在 DOUT 端。数据在 CLK 的下降沿输出。数据标记为 D15~D0。D11~D8 为寄存器地址, D7~D0 为数据, D15~D12 为“任意”位。接收到的第一位为 D15, 是最高位(MSB)。

2. 数字和控制寄存器

表 7-16 列出了 14 个可寻址数字和控制寄存器, 数字寄存器由一个片内 8×8 双端口 SRAM 组成。控制寄存器包括译码方式、显示亮度、扫描界线(扫描数据字的数量)、掉电和显示测试。

表 7-16 可寻址数字和控制寄存器

寄存器	地址					十六进制代码
	D15~D12	D11	D10	D9	D8	
空操作	0	0	0	0	0	00H
DIG0	0	0	0	0	1	01H
DIG1	0	0	0	1	0	02H
DIG2	0	0	0	1	1	03H
DIG3	0	0	1	0	0	04H
DIG4	0	0	1	0	1	05H
DIG5	0	0	1	1	0	06H
DIG6	0	0	1	1	1	07H

续表

寄存器	地 址					十六进制代码
	D15~D12	D11	D10	D9	D8	
DIG7	0	1	0	0	0	08H
译码方式	0	1	0	0	1	09H
显示亮度	0	1	0	1	0	0AH
扫描界限	0	1	0	1	1	0BH
掉电	0	1	1	0	0	0CH
显示测试	0	1	1	1	1	0FH

1) 掉电方式

MAX7219 工作于掉电方式时, 扫描振荡器停止工作, 此时将所有段电流源接地, 所有的数字驱动器被拉到 V_{cc} , 显示器不显示。在数据和控制寄存器中的数据保持不变。停机能用来节电, 也可通过依次进入或离开掉电模式或闪烁显示作为一个警告。在掉电模式为最小供给电流时, 逻辑输入应接地或接 V_{cc} (CMOS 逻辑电平)。

2) 起始上电

在起始上电时, 所有控制寄存器被复位, 显示器不显示, 并且 MAX7219 进入掉电方式。在正常显示前, 必须先给显示驱动器初始化。否则, 它将被设置成扫描一个数字, 而且它 will 不译码数据寄存器中的数据, 并且亮度寄存器的亮度被设置为最小。

3) 译码方式寄存器

译码方式寄存器对每个数字设置 BCD 码(0~9、E、H、L、P 和-)或非代码操作。寄存器中的每一位与一个数字相对应。逻辑高电平选择 BCD 译码, 而逻辑低电平不译码。

当采用 BCD 译码方式时, 译码器仅针对数字寄存器中数据的低四位(D3~D0), 而不考虑 D4~D6 位。设置小数点的 D7 与译码器无关, 且为正逻辑(D7 = 1 时接通小数点)。

BCD 码字形是: 当数据为 00H~09H 时, 显示 0~9; 当数据为 0AH~0EH 时, 显示“-”, E, H, L, P”。

当选择不译码方式时, 数据位 D7~D0 对应于 MAX7219 的段线。每一个数据位同时对应段线相连的情况如图 7-9 所示。

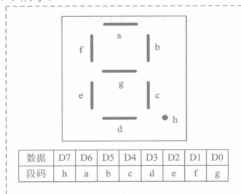


图 7-9 LED 字段与数据的对应关系

4) 显示亮度

MAX7219 允许通过外接电阻(Rset)控制显示亮度。段驱动器的峰值电流刚好是通过 Rset 电流的 100 倍。此寄存器的内容也能调节(改变)亮度。一般设置段电流为 40 mA, 它的最小值应该是 9.53 Ω 。显示亮度也可以通过使用亮度寄存器来进行数字控制。

显示亮度的数字由一个内部的脉宽调制器提供, 它通过亮度寄存器的低四位来控制。调节器分 16 等级, 把由 Rset 设置的峰值电流从最大的 31/32 降到 1/32 来确定段电流的平均值。

5) 扫描界限寄存器

扫描界限寄存器设置所显示数据的多少, 可为 1~8。它们一般以扫描速率 800 Hz、8 位数据、多路复用方式显示。如果显示的数据较少, 则扫描速率为 $8 \times f/N$, 其中 N 为扫描数字的数量, f 为扫描频率。既然所扫描数字的数量影响显示亮度, 那么扫描界限寄存器就不应该再用来显示空白位(例如, 禁止开头的零显示)。扫描界限寄存器的格式见表 7-17。

表 7-17 扫描界限寄存器格式(地址 = 0BH)

扫描界限	寄存器数据								十六进制码
	D7	D6	D5	D4	D3	D2	D1	D0	
仅显示位 0	0	0	0	0	0	0	0	0	00H
显示位 0 和 1	0	0	0	0	0	0	0	1	01H
显示位 0、1 和 2	0	0	0	0	0	0	1	0	02H
显示位 0、1、2 和 3	0	0	0	0	0	0	1	1	03H
显示位 0、1、2、3 和 4	0	0	0	0	0	1	0	0	04H
显示位 0、1、2、3、5 和 5	0	0	0	0	0	1	0	1	05H
显示位 0、1、2、3、4、5 和 6	0	0	0	0	0	1	1	0	06H
显示位 0、1、2、3、4、5、6 和 7	0	0	0	0	0	1	1	1	07H

如果扫描界限寄存器被设置为 3 个数字或更少, 则各个数字驱动器将消耗过量的功率。因此, Rset 电阻值必须调节到和位显示器数目相一致, 以限制各个数字驱动器的功率消耗。

6) 显示测试寄存器

显示测试寄存器有两种工作方式: 正常和显示测试。显示测试方式不改变所有控制和数字寄存器(包括停机寄存器)来接通所有 LED。在显示测试方式下, 8 位数字被扫描, 占空比为 31/32。当在 0FH 中送 01H 时, 为测试; 当送 00H 时, 为正常。

注意: 一旦 MAX7219 设为测试方式(所有 LED 全亮), 则一直保持到显示测试寄存器被置为正常工作方式为止。

7) 非工作寄存器

当 MAX7219 级联时, 使用非工作寄存器, 把所有器件的 LOAD 输入连接在一起, 而把 DOUT 连接到相邻 MAX7219 的 DIN 上。DOUT 为 CMOS 逻辑电平输出, 易于依次级联 MAX7219 的 DIN。如果 4 片 MAX7219 级联, 那么对第 4 片芯片写入时, 发送所需的 16 位字, 其后限有三个非工作代码, 当 LOAD 变高时数据被锁存在所有器件中。前三个芯片接收非工作指令, 而第 4 个芯片接收预期的数据。

8) 电源旁路及布线

要使由峰值数字驱动器电流引起的纹波减到最小,需在 V_{cc} 到 GND 间尽可能靠近芯片,外接一个 $10\ \mu\text{F}$ 的电解电容和一个 $0.1\ \mu\text{F}$ 的陶瓷电容。MAX7219 应放置在靠近 LED 显示器的地方,保证对外引线尽量短,以减小引线电感和电磁干扰。

9) 选择 Rset 电阻和使用外部驱动器

MAX7219 的最大推荐电流是 $40\ \text{mA}$ 。当段电流超过此标准时,需要使用外部的数字驱动器。因此,当使用外部电流源作为段驱动器时,使用 $R_{\text{set}} = 47\ \text{k}\Omega$ 来达到节能的目的。

7.2.2 应用电路与编程

图 7-10 为 MAX7219 组成的 8 位 LED 数字显示电路。单片机的 I/O 口 P1.0 和 P1.2 分别作为 MAX7219 的串行数据输入信号 DIN 和时钟信号 CLK, P1.1 作为 LOAD 信号。

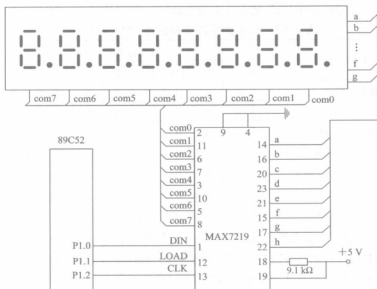


图 7-10 MAX7219 构成的 8 位 LED 数字显示电路

用 C51 完成的源程序如下:

```
#include <reg52.h>
sbit din_dsp=P1^0;          /*定义 DIN 引脚*/
sbit clk_dsp=P1^2;          /*定义 CLK 引脚*/
sbit load_dsp=P1^1;         /*定义 LOAD 引脚*/

/*数码管数据: 0,1,2,3,4,5,6,7,8,9,-,E,H,L,P, 常驻数据定义在程序区*/
unsigned char code show_num[]={0x7e,0x30,0x6d,0x79,0x33,0x5b,0x5f,0x70,
                                0x7f,0x7b,0x01,0x4f,0x37,0x0e,0x67,0x00,0x37,0x0e};

void delay(unsigned char k)  /*短延时函数, k 是延时常数*/
{
    unsigned char i;
```

```

        for (i=0; i<k; i++);
    }

/*MAX7219 底层送 16 位数据, address 是地址, numb 是数据*/
void send_7219(unsigned char address,unsigned char numb)
{
    unsigned char adds,led,i,j;
    adds=address; led=numb; j=numb;
    load_dsp=1; clk_dsp=0;
    delay(10); load_dsp=0; /*LOAD 变低*/
    for(i=0; i<8; i++) /*送地址*/
    {
        j=adds&0x80;
        if(j) din_dsp=1; /*先从高位开始*/
        else din_dsp=0;
        clk_dsp=1; delay(5);
        clk_dsp=0; delay(5);
        adds=adds<<1; /*左移 1 位*/
    }
    for(i=0; i<8; i++) /*送 led 数据*/
    {
        j=led&0x80;
        if(j) din_dsp=1;
        else din_dsp=0;
        clk_dsp=1; delay(5);
        clk_dsp=0; delay(5);
        led=led<<1; /*左移 1 位*/
    }
    load_dsp=1; /*LOAD 置高*/
}

/*显示函数*/
void disp_7219(unsigned char add,unsigned char ch)
{
    send_7219(add,show_num[ch]);
}

/*MAX7219 初始化函数*/
void set_max7219(void) /*初始化 MAX7219*/
{
    send_7219(0x0c,0x01); /*正常显示方式*/
    send_7219(0x09,0x00); /*采用不译码方式(用代码方式)*/
}

```

```

send_7219(0x0f,0x00);    /*正常操作模式*/
send_7219(0x0a,0x01);    /*显示亮度为 8(0~F)*/
send_7219(0x0b,0x07);    /*8 位显示*/
}

/*显示器清零函数*/
void clr_7219(void)
{
    unsigned char i;
    for(i=1; i<=8; i++)
        send_7219(i,0x00);
}

/*带小数点显示函数*/
void disp_dp(unsigned char add,unsigned char ch)
{
    send_7219(add,show_num[ch]|0x80);
}

void main(void)           /*测试显示 0~7 八个字*/
{
    unsigned char i;
    set_max7219();         /*初始化*/
    clr_7219();            /*显示器清零*/
    for(i=0; i<8; i++)
        disp_7219(i,i);    /*显示 0~7*/
    while(1);              /*等待*/
}

```

7.3 SLE4432/42 接触式加密存储卡

7.3.1 硬件与功能描述

IC 卡(Integrated Circuit Card)按电路形式可分为三大类:存储器卡、逻辑加密卡和 CPU 卡。存储器卡价格低廉,操作简单,但安全性差,应用范围窄;CPU 卡安全性高,但价格也高,且对配套设备要求高,故主要应用于安全性很高的场合;逻辑加密卡安全性较高,价格适中,故在各类系统中得到了广泛应用。逻辑加密卡种类很多,其中 SLE4432/42 是目前应用最普遍的一种逻辑加密卡。SLE4432/42 采用 I²C 总线,其触点配置和接口标准符合 ISO7816 同步传输协议,采用 NMOS 工艺制造。

1. 主要性能特点

- (1) 存储空间: 256 × 8 位 E²PROM;
- (2) 写保护: 最低 32(0~31)字节,具有不可改的写保护,其结构是 32 × 1 位的保护存

存储器:

- (3) 每字节编程时间: 2.5 ms(典型值);
- (4) 擦写次数: >1 万次;
- (5) 数据保存时间: >10 年;
- (6) 供电: +5 V \pm 10%;
- (7) 工作电流: <10 mA;
- (8) 工作温度: -45~+85℃(工业级)。

2. 内部结构与引脚说明

SLE4432/42 的内部结构与引脚排列如图 7-11 所示。

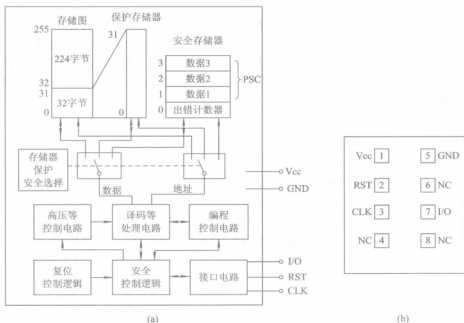


图 7-11 SLE4432/42 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

SLE4432/42 的引脚排列如图 7-11(b)所示。其中:

- (1) Vcc 是电源端, 一般接+5 V(最大为+6 V);
- (2) RST 是复位输入端, 高电平有效(最小 5 μ s);
- (3) CLK 是同步时钟输入端(7~50 kHz);
- (4) I/O 是数据的输入/输出端;
- (5) NC 悬空, 不用;
- (6) GND 是器件参考地。

3. SLE4432/42 的异同点

1) SLE4432

SLE4432 的 2K(256 字节)位存储卡电路具有一个 256 字节的 E²RPOM 主存储区和一个

32 位 PROM 保护存储区。主存储区按字节进行删除和写入。删除时, 数据字节的 8 位都置为逻辑 1; 写入时, E^2 PROM 各单元的信息可按输入数据一位一位改写成逻辑 0(E^2 PROM 中的新旧数据进行逻辑与)。通常, 一次数据的改写过程由一次删除和一次写入过程组成。 E^2 PROM 单元是否真的被删除和写入, 取决于主存储区中数据字节和新的数据字节的内容。如果所定位字节的 8 位没有一位需要 0 至 1 翻转, 则跳过删除操作; 如果不需要 1 至 0 的翻转, 则省去了写入操作。

起始的 32 个字节可以通过写保护存储区中对应的位而不可逆转地防止被改写。该地址范围内的每个数据字节与保护存储区中的一位相对应, 而且与主存储器具有相同的地址。保护位一旦写入就不能再删除。

2) SLE4442

SLE4442 的 2K 位(256 字节)加密存储卡电路除具有上述 SLE4432 的功能外, 还提供一个控制对存储区进行写入/删除操作的密码逻辑, 为此 SLE4442 包含一个出错计数器。加电后, 除这些参考数据外, 整个存储区只能被读取。只有在校验数据和内部参考数据比较相同后才可能进行写入和删除操作。在三次比较失败时, 出错计数器就封锁所有后续的任何操作。

4. 操作说明

1) 传输协议

传输协议为接口设备和 IC 卡之间的 2 线连接协议。协议类型标识为 S=10。I/O 上的所有数据交换由 CLK 的下降沿触发。

传输协议由复位和响应复位、命令模式、数据输出模式和处理模式 4 个部分组成。

(1) 复位和响应复位。响应复位按 ISO7816-3 标准产生。在操作期间, 任何时候都可以给出复位信号。复位时, 地址计数器由一时钟脉冲置到零。在 RST 从高电平(H 态)置成低电平(L 态)时, 第一个数据位(LSB)输出到 I/O。通过此后连续 31 个时钟脉冲, 可读出前 4 个 E^2 PROM 地址单元中的内容。第 33 个时钟脉冲将 I/O 置成 H 态, 如图 7-12(a)所示。在响应复位期间, 将忽略所有启动和停止条件。

(2) 命令模式。响应复位后, IC 卡等待命令的输入。每条命令从一启动条件开始, 包括 3 个字节长的命令体及其后的一个附加时钟脉冲, 最后由停止条件结束, 如图 7-12(b)所示。

启动条件: CLK 处于高电平时(H 态期间), I/O 线由高到低变化(下降沿)。

停止条件: CLK 处于高电平时(H 态期间), I/O 线由低到高变化(上升沿)。

接收命令后, IC 卡有两种可能的模式: 读操作时的数据输出模式及写入和删除操作时的处理模式。

(3) 数据输出模式。在 CLK 上的第一个下降沿后, I/O 上第一个数据位有效。在最后一个数据位后, 为使 I/O 成为 H 态并使 IC 卡准备好接收新命令, 需要一个额外的时钟脉冲。在此模式期间, 任何起始和停止条件均不起作用。该模式的时序如图 7-12(c)所示。

(4) 处理模式。在此模式下, IC 卡进行内部处理, 图 7-12(d)是其时序图。在 CLK 的第一个下降沿后变成 L 态的 I/O 线恢复 H 态前, 必须向 IC 连续提供 m 个时钟信号。在此模式期间, 任何起始和停止条件均不起作用。

2) 命令

(1) 命令格式。SLE4432/42 存储卡的命令格式为由三个字节组成, 如表 7-18 所示。

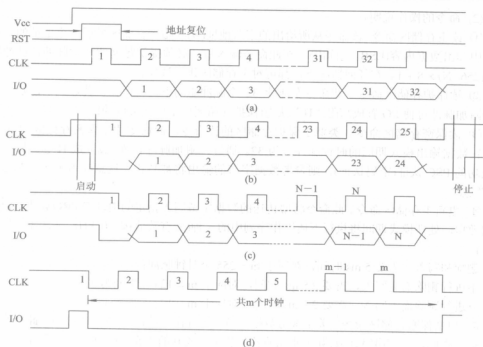


图 7-12 SLE4432/42 的传输时序

(a) 复位时序; (b) 命令模式; (c) 数据输出模式; (d) 处理模式时序

表 7-18 命令的组成

控制字节(8 位)	地址字节(8 位)	数据字节(8 位)
B7~B0	A7~A0	D7~D0
数据传输顺序由 B0 开始, 即 B0~B7、A0~A7、D0~D7(共 24 位数据)		

SLE4432/42 IC 存储卡的命令由控制字节决定, 其中 SLE4432 有 4 条, SLE4442 附加了 3 条, 见表 7-19。

表 7-19 SLE4432/42 的命令格式

SLE4432/42 卡, 共 4 条											
控制字节				地址字节	数据字节	操作说明	模式说明				
B7	B6	B5	B4	B3	B2			B1	B0	A7~A0	D7~D0
0	0	1	1	0	0	0	0	地址	—	读主存储区	数据输出
0	0	1	1	1	0	0	0	地址	输入数据	更新主存区	处理
0	0	1	1	0	1	0	0	—	—	读保护区	数据输出
0	0	1	1	1	1	0	0	地址	输入数据	写保护区	处理
SLE4442 卡, 共 3 条											
0	0	1	1	0	0	0	1	—	—	读保密区	数据输出
0	0	1	1	1	0	0	1	地址	输入数据	更新保密区	处理
0	0	1	1	0	0	1	1	地址	输入数据	比较验证	处理

(2) 命令的操作说明。

① 读主存储区命令。该命令从所给出的字节地址(N)开始到存储区最后一个地址的主存储区中读出数据内容(LSB 先读出)。在此命令输入后,必须提供足够的时钟脉冲,脉冲数 $m = (256 - N) \times 8 + 1$ 。在任何情况下,都可对主存储区进行读操作。

② 读保护存储区命令。此命令在连续 32 个脉冲驱动下,将保护位传送到输出端,利用一附加脉冲可使 I/O 置成高阻“H”状态(保护存储区还是可以读的)。

③ 读保密存储区命令。类似于保护存储区的读命令,此命令读出保密存储区的 4 个字节:数据输出模式期间的时钟脉冲数为 32。通过一附加脉冲, I/O 被置成 H 态。如无预先成功的 PSC 认证(密码认证),则参考数据字节的输出将被禁止。这意味着 I/O 仍然处在“L”态。

④ 更新主存储区命令。此命令将要传送的数据字节写入指定的地址 E^2 PROM 中。根据新旧数据,处理模式期间将执行下述操作序列中的一种(所有数值按 50 kHz 时钟速率计算所得):

删除和写入,需要 5 ms 时间,对应于 $m = 255$ 个时钟脉冲;

不进行删除而无写入,需要 2.5 ms 时间,对应于 $m = 124$ 个时钟脉冲;

只进行删除而无写入,需要 2.5 ms 时间,对应于 $m = 124$ 个时钟脉冲。

⑤ 更新保密存储区命令。关于参考数据字节,只有在成功地认证了 PSC 后,此命令才能执行,否则,只有出错计数器(地址 0)的各位从 1 写成 0 的更新,执行时间和所需的时钟脉冲与上述“更新主存储区”时相同。

⑥ 写保护存储区命令。此命令的执行包括一个输入数据字节与 E^2 PROM 中指定字节的比较过程,当对应在写保护位已写入时,如两者一致,则执行写保护位操作,使数据信息不可改变。如果数据比较结果,两者不同,则写保护位的操作将被禁止。执行时间和所需脉冲见“更新主存储区命令”。

⑦ 比较认证数据命令。此命令只能与出错计数器的一次更新步骤组合使用(见比较命令用法),该命令将输入的一个认证数据字节和对应的参考数据字节进行比较,对此过程来说,处理模式期间的时钟脉冲是必需的。

⑧ 比较命令的用法。这种方法必须完全按下述步骤执行。任何变更将带来失败,以致写/擦均无法进行。只要此过程不能成功结束,出错计数器各位就只能从 1 变为 0,而不能进行删除。首先,出错计数器的某一位必须通过一条更新命令写成 0,如图 7-13 所示。然后,从参考数据的字节 1 开始执行三次“比较认证命令”。至此,只要有操作电压,就可以对整个存储区域进行写入(删除)操作。假若出错,只要还有出错计数器位,上述整个过程就可以进行下去。在已经许可的情况下,参考数据可以像 E^2 PROM 中的任何其他信息一样被改变。在器件出厂时, PSC 的设置可根据用户要求独立进行编码。这样,要改变此数据就必须知道此代码(密码)。

3) 复位模式

复位和响应复位是相对于传输协议而言的。

(1) 复位时的供电。在将 V_{cc} 与操作电压接通后, I/O 处于高态(H)。在可以改变数据之前,必须执行一次对任意地址或响应复位的读操作。

(2) 中止。如果 CLK 处于 L 态时将 RST 置 H 态,则任何操作都将被中止,同时 I/O 被

置成 H 态。为触发一次已定义的有效复位, RST 的高电平宽度需要 $5\mu\text{s}$ 的最短间隔。在中止后, IC 卡等待进一步的操作。

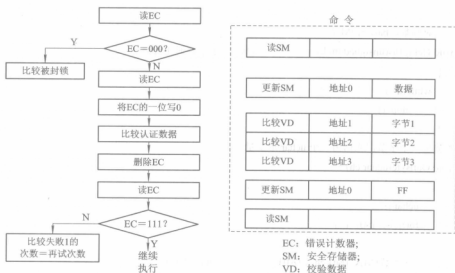


图 7-13 认证过程

7.3.2 编程方法

/*硬件采用 STC89C52RD 单片机(12 MHz 晶振), 管脚定义见下列程序*/

```

#include <stdio.h>
#include <intrins.h>
#include <reg52.h>
typedef unsigned char uchar;
typedef unsigned int uint;
#define RMM_COMM 0x30 /*读主存命令字*/
#define UMM_COMM 0x38 /*写主存命令字*/
#define CVD_COMM 0x33 /*校验密码*/
#define RSM_COMM 0x31 /*读密码存储区*/
#define USM_COMM 0x39 /*写密码存储区*/
#define RPM_COMM 0x34 /*读保护存储区*/
#define WPM_COMM 0x3c /*写保护存储区*/
#define Pow_On 0 /*低电平供电 sw 开关*/
#define Pow_Off 1 /*高电平断电*/
/*定义 IC 卡接口*/
sbit CLK=P3^6; /*时钟*/
sbit OUTDATA=P1^3; /*输出(CPU 侧)*/
sbit INDATA=P1^3; /*输入(CPU 侧)*/

```

```

sbit POWER=P3^3;          /*电源(SW)*/
sbit RST=P2^0;             /*IC 卡复位脚*/

/*延时函数, base=12M*/
void Delay1us(unsigned int k)    /*k=10 表示延时 10 μs*/
{
    while(k--)
        _nop_();
}

/*输出一个字节, 入口为一个 uchar 形变量*/
void OutByte(uchar ch)
{
    uchar i;
    for(i=8; i>0; i--)
    {
        OUTDATA=(bit)(ch & 0x01);    /*强制取为位变量, 输出*/
        Delay1us(10); CLK=1;
        Delay1us(10); CLK=0;
        ch=ch>>1;                    /*右移 1 位*/
    }
}

uchar InByte(void)             /*从卡读一个 8 位数据*/
{
    uchar i,a=0;
    INDATA=1;                  /*置为输出状态*/
    for(i=8; i>0; i--)
    {
        a=a>>1;
        CLK=1; Delay1us(10);
        if(INDATA==1) a|=0x80;
        CLK=0; Delay1us(10);
    }
    return(a);
}

/*结束命令模式, 没有入口*/
void Stop_Comm(void)
{
    OUTDATA=0;
    Delay1us(10); CLK=1;
}

```

```

    Delay1us(10); OUTDATA=1;
    Delay1us(10);
}

void Start_Comm(void) /*启动函数*/
{
    OUTDATA=1; Delay1us(6);
    CLK=1; Delay1us(10);
    OUTDATA=0; Delay1us(10);
    CLK=0; Delay1us(5);
}

/*处理模式。开始处理模式后，卡片将输入口拉低，然后等待输入口变成高电平*/
/*如果等待超过 10 ms(BASE=12 MHz)，则程序退出*/
void Proce_Mod(void)
{
    uint i;
    CLK=0; Delay1us(10);
    OUTDATA=1; INDATA=0;
    for(i=620; i>0; i--)
    {
        CLK=1; Delay1us(10);
        CLK=0; Delay1us(10);
        if(INDATA) break; /*输入为高就跳出*/
    }
}

/*发送命令。入口：a 为命令字，b 为地址，c 为数据*/
void SendComm(uchar a,uchar b,uchar c)
{
    Start_Comm(); /*开始发送命令*/
    OutByte(a); /*发命令字*/
    OutByte(b); /*发地址*/
    OutByte(c); /*发数据*/
    Stop_Comm(); /*结束发送命令*/
}

/*卡复位，用于结束读模式*/
void RstCard(void)
{
    uchar temp;
    RST=1; Delay1us(10);
    CLK=1; Delay1us(20);

```

```

        CLK=0; Delay1us(10);
        RST=0; Delay1us(10);          /*发出复位时序*/
        temp=InByte(); temp=InByte();
        temp=InByte(); temp=InByte();  /*空读 4 次*/
    }
    /*连续输入 i(<=255)个字节, 存放到以 pt 开头的内部单元中*/
    /*必须在某一读数据命令模式之后使用*/
    /*入口: pt 为起始地址, i 为数据个数*/
    void Read_Mod(uchar idata *pt,uchar i)
    {
        CLK=0; Delay1us(10);
        do
        {
            *pt=InByte();          /*读入一个字节*/
            pt++;                  /*指针加 1*/
        }
        while(--i);               /*计数器减 1, 并判断*/
    }
    /*写 IC 卡主存, 每次写 1 字节*/
    /*入口: IC 卡地址, 指向数据区的指针*/
    void Umm(uchar CardAdd,uchar *pt)
    {
        SendComm(UMM_COMM,CardAdd,*pt);  /*写主存的命令字、地址和数据*/
        Proce_Mod();
    }
    /*读 IC 卡主存*/
    /*入口: 卡地址、指向内部 RAM 的指针和数量*/
    void Rmm(uchar cardAdd,uchar *pt,uchar i)
    {
        SendComm(RMM_COMM,cardAdd,0);
        Read_Mod(pt,i);
        RstCard();
    }
    /*读保护存储器*/
    /*入口: 指向直接寻址数据区的指针*/
    void Rpm(uchar *pt)
    {
        SendComm(RPM_COMM,1,1);          /*读保护存储器的命令字, 后两个参数忽略*/
        Read_Mod(pt,4);                  /*读出 4 个字节*/
    }

```

```

/*保护一字节,注意待保护的字节是已经写入过的,地址只能在保护存储区内*/
/*入口: IC 卡地址,指向直接寻址数据区的指针*/
void P_Byte(uchar CardAdd,uchar *pt)
{
    SendComm(WPM_COMM,CardAdd,*pt); /*写主存的命令字、地址和数据*/
    Proce_Mod();
}

/*卡上电,延时约 80  $\mu$ s,卡复位,同时读入 4 个标志字节*/
/*入口: 指向存放标志的内部直接寻址 RAM 地址*/
void Power_On(uchar *pt)
{
    POWER=Pow_On; /*上电*/
    INDATA=1;
    OUTDATA=1;
    Delay1us(80); /*延时 80  $\mu$ s*/
    RST=1; Delay1us(10);
    CLK=1; Delay1us(20);
    CLK=0; Delay1us(10);
    RST=0; Delay1us(10); /*复位*/
    i=4;
    do
    {
        *pt=InByte();
        pt++;
    }
    while(--i); /*读入 4 个标记字节*/
}

/*校验密码*/
/*入口: 指向存放密码的内部直接寻址 RAM 地址*/
/*出口: 成功返回 1,失败返回 0,若卡片已锁,则也返回 0*/
bit Verify(uchar *pt)
{
    uchar temp[4]; /*暂存 4 字节的保密区内容*/
    uchar *tpt;
    uchar i;
    tpt=&temp[0];
    SendComm(RSM_COMM,1,1) /*读保密存储区的命令字*/
    Read_Mod(tpt,4); /*读出 4 字节*/
    if((temp[0] & 0x07)!=0) /*第一字节是重试计数器,当不全为 0 时*/
    {

```

```

    if((temp[0]&0x07)==0x07) i=0x06;
    else if((temp[0]&0x07)==0x06) i=0x04;
    else if((temp[0]&0x07)==0x04) i=0x00; /*将其中一位为1的改为0*/
    SendComm(USM_COMM,0,i); /*重写计数器*/
    Proce_Mod(); /*处理*/
    for(i=1; i<4; i++,pt++) /*校对3字节的密码*/
    {
        SendComm(CVD_COMM,i,*pt); /*发出校对命令*/
        Proce_Mod(); /*处理*/
    }
    SendComm(USM_COMM,0,0xff); /*擦除计数器*/
    Proce_Mod(); /*处理*/
    SendComm(RSM_COMM,1,1); /*读保密存储区的命令字*/
    tmp=&temp; /*写到数组*/
    Read_Mod(tmp,4); /*读出计数器的内容*/
    if((temp[0]&0x07)==0x07) /*如果没有被成功擦除,则校对失败*/
        return(1);
    }
    return(0);
}

void main(void) /*测试程序部分*/
{
    uchar idata *pt;
    unsigned int i;
    bit temp_flag;
    uchar idata test[10]={1,2,3,4,5,6,7,8,9};
    uchar idata test1[50];
    for(i=0; i<50; i++)
        test1[i]=0x00;
    pt=&test1[0]; /*密码*/
    RstCard(); /*复位函数*/
    Power_On(pt); /*上电同时读入标志字节*/
    Delay1us(1000); /*延时*/
    *pt=0xff; /*第一个密码*/
    pt++; *pt=0xff; /*第二个密码*/
    pt++; *pt=0xff; /*第三个密码,三个密码为0xff、0xff、0xff*/
    pt=&test1[0];
    temp_flag=Verify(pt); /*校验密码*/
    if(temp_flag==1) /*校验成功*/

```

```

    {
        pt=&test1[0];
        Rpm(pt);                /*读入保密区字节*/
        *pt=0xaa;               /*写主存*/
        Umm(0x04,pt);           /*保护 0x05 地址的数据*/
        P_Byte(0x05,pt);
        Umm(0x20,0x03);
        Delay1us(50);
        pt=&test1[0];
        Rmm(0x20,pt,50);        /*读主存*/
        for(i=0x20; i<(0x20+50); i++)
        { printf("%c",pt[i-0x20]); /*输出*/
            Delay1us(10);
        }
    }
    while (1);                 /*等待*/
}

```

7.4 MCP6S21/2/6/8 可编程增益模拟放大器

7.4.1 硬件与功能描述

MCP6S21/2/6/8 是可编程增益模拟放大器(PGA)。通过 SPI 接口可配置 8 个输出增益和选择 8 个通道中的一个放大通道。通过串行接口设置可以将 PGA 置为关断模式,以降低功耗。这些 PGA 针对高速度、低失调电压和单电源操作进行了电路优化,具有轨到轨输入和输出能力。该器件支持更加灵活的多输入或单输入模拟信号放大。

MCP6S21、MCP6S22、MCP6S26 和 MCP6S28 分别是单通道、双通道、六通道和八通道放大器。该系列器件可广泛应用于 A/D 转换器的驱动器、复用的模拟放大器、数据采集、工业仪表、测试设备和医疗仪器等系统。

1. 主要性能特点

- (1) 复用输入: 1、2、6 或 8 通道;
- (2) 8 种增益选择: +1、+2、+4、+5、+8、+10、+16 和+32;
- (3) 串行 SPI 接口;
- (4) 轨到轨输入和输出模式;
- (5) 增益误差: $\pm 1\%$ (最大值);
- (6) 失调电压低: $\pm 275 \mu\text{V}$ (最大值);
- (7) 带宽: 2~12 MHz(典型值);
- (8) 噪声低: 10 kHz 时为 $10 \text{ nV}/\sqrt{\text{Hz}}$ (典型值);
- (9) 供电电流: 1.0 mA(典型值);

(10) 单电源供电: 2.5~5.5 V;

(11) 工作温度: -40~+85℃。

2. 内部组成与引脚说明

MCP6S21/2/6/8 可编程增益模拟放大器的内部组成与引脚排列如图 7-14 所示。

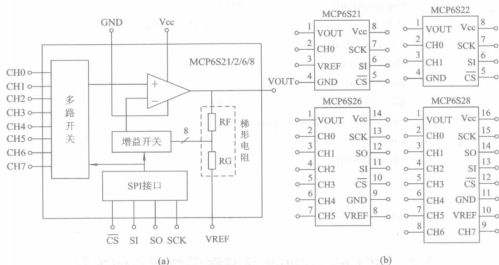


图 7-14 MCP6S21/2/6/8 的内部组成与引脚排列

(a) 内部组成; (b) 4 种器件的引脚排列

MCP6S21/2/6/8 的引脚排列如图 7-14(b)所示。其外形封装有 DIP-8、SOIC-8、MSOP-8、DIP-14、SOIC-14、SSOP-14、DIP-16 和 SOIC-16 等几种。其中:

(1) CH7~CH0 是模拟信号输入端。输入信道与型号有关, 最少 1 个通道, 最多 8 个通道。它是高阻抗、低偏置电流的 CMOS 输入。内部通过多路开关选择至放大的同相输入端。

(2) VOUT 是放大器输出端, 是低阻抗电压源。其输出受 CH7~CH0 模拟信号、内部网络电阻、VREF 等控制。

(3) VREF 是外部参考电压输入端, VREF 引脚的电压应介于 GND 和 Vcc(MCP6S22 的 VREF 内部连接到 GND)。该引脚上的电压将会平移输出电压。

(4) Vcc、GND 是器件电源输入端。正电源引脚(Vcc)的电压比地(GND)高 2.5~5.5 V。其他引脚的电压应介于 GND 和 Vcc 之间以便正常工作。通常, 这些器件用在单(正)电源配置中。在这种情况下, GND 就是地线, Vcc 接到电源。同时需要在 Vcc 引脚上就近接旁路电容(0.1 μF), 它可以和 Vcc 引脚附近的模拟器件共用大容量电容(通常为 2.2~10 μF)。

(5) SCK 是 SPI 接口时钟输入端。

(6) SI 是 SPI 接口的数据输入端。

(7) CS 是器件数字接口片选输入端, 低电平有效。

(8) SO 是 SPI 接口的数据输出端(只有 MCP6S2/6/8 有)。它是 CMOS 推挽式输出, 平时为高阻抗。一旦 CS 变为高电平(未选中), SO 将被强制置为低电平。该功能支持菊花链连接。

3. 模拟部分

内部运算放大器提供合适的带宽、精度和增益。

1) 补偿电容

内部运算放大器有三个连接到交换网络的补偿电容。选择它们可使在高增益下提供良好的小信号带宽，在低增益下提供良好的压摆率(全功率带宽)。随增益变化而发生的带宽变化介于 2~12 MHz 之间，见表 7-20。

表 7-20 内部增益补偿电容($V_{CC} = 5\text{ V}$ 时测试)

增益 /(V/V)	内部补偿 程度电容	典型 GBWP /MHz	典型 SR /(V/ μs)	典型 FBPW /MHz	典型 BW /MHz
1	大	12	4.0	0.30	12
2	大	12	4.0	0.30	6
4	中	20	11	0.70	10
5	中	20	11	0.70	7
8	中	20	11	0.70	2.4
10	中	20	11	0.70	2.0
16	小	64	22	1.6	5
32	小	64	22	1.6	2.0

2) 轨到轨输入/输出

内部运算放大器的输入级采用两个并联的差分输入级，一个工作于低 V_{IN} (输入电压)，另一个工作于高 V_{IN} 。使用这种拓扑结构，内部输入可工作在超出供电轨 0.3 V 的状态下。输入失调电压在 $V_{IN} = 0 \sim 0.3\text{ V}$ 到 $V_{CC} + 0.3\text{ V}$ 均进行了测量，以确保正常工作。 $V_{IN} \approx V_{CC} - 1.5\text{ V}$ 时，两个输入级之间会发生跃迁。为获得最佳的畸变和增益线性度，应避免在这个区域工作。

最大输出电压是特定输出负载下的最大摆幅。当 $R_L = 10\text{ k}\Omega$ 且 $V_{REF} = V_{CC}/2$ 时，输出可达到任一供电轨的 60 mV 内。

3) 输入电压和相位翻转

系列放大器的设计采用 CMOS 输入器件。其技术保证了输入引脚超出供电电压时不会引起相位翻转。可加在输入引脚(CHX)上的最大电压是 $0 \sim 0.3\text{ V}$ 到 $V_{CC} + 0.3\text{ V}$ 。超出该绝对极限参数值的输入电压可能导致流入或流出输入引脚的电流过大。若超过 $\pm 2\text{ mA}$ 的电流可能导致可靠性问题，则对于超过该参数的应用，必须使用输入电阻在外部进行限流。

4) 梯形电阻

图 7-14(a)所示的梯形电阻($RLAD = R_F + R_G$)用于设定增益。将增益开关与反相输入串联，可减小寄生电容、畸变和增益失调。 $RLAD$ 是 PGA 输出的额外负载。

在关断模式下， $RLAD$ 仍连到 V_{OUT} 和 V_{REF} 引脚。因此， V_{OUT} 和 V_{REF} 引脚以及内部放大器的反向输入都通过 $RLAD$ 连接在一起，并且输出不是高阻抗(不同于外部运算放大器)。尽管 $RLAD$ 会增加输出噪声，但其影响很小。

5) 关断模式

SPI 接口发出关断命令时, 内部运算放大器被关断, 其输出置于高阻抗状态。由于梯形电阻的存在, 输出电阻约为 $5\text{ k}\Omega$, 并且存在一条将输出信号馈送到输入的路径。激活上电复位电路时, 将该器件暂时置于关断状态。

4. 数字部分

MCP6S21/2/6/8 PGA 使用标准的 SPI(兼容串行接口)接收来自单片机的指令。该接口的配置允许和其他 SPI 器件形成菊花链。有一个内部上电复位可将寄存器复位到低功耗状态。

1) 时序

片选($\overline{\text{CS}}$)翻转为低电平, 可以启动与这些器件的通信。每个 SI 字(双字节长)的第一个字节是指令字节, 进入指令寄存器后, 指令寄存器将第二个字节指向其目标单元。在典型的应用中, $\overline{\text{CS}}$ 在一个字(16 位)后变为高电平。

器件工作于 SPI 的 00 模式和 11 模式。在 00 模式下, 时钟在低电平状态空闲; 在 11 模式下, 时钟在高电平状态空闲。在这两种模式下, SI 数据都在 SCK 的上升沿装入 PGA, 而 SO 数据在 SCK 的下降沿送出。在 00 模式下, $\overline{\text{CS}}$ 的下降沿也用作 SCK 的第一个下降沿。当 $\overline{\text{CS}}$ 为低电平时, 时钟周期(SCK)数必须为 16 的倍数, 否则命令将中止。时序如图 7-15 所示。

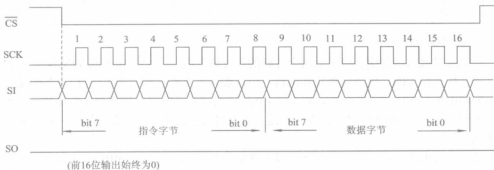


图 7-15 PGA 操作时序

2) 寄存器设置

放大器通过 SPI 接口设置功能。数据发送到三个 8 位寄存器(指令寄存器、增益寄存器和通道寄存器)中的两个。

(1) 指令寄存器。指令寄存器的格式如表 7-21 所示。

表 7-21 指令寄存器的格式(默认值: 000xxxx0)

D7	D6	D5	D4	D3	D2	D1	D0
可写位	可写位	可写位	未用	未用	未用	未用	可写位
M2	M1	M0	—	—	—	—	A0

表 7-21 中: A0 是间接寻址位。A0 = 1 是寻址通道寄存器, A0 = 0 是寻址增益寄存器(默认)。

M2、M1、M0 是命令位。其中, 当 M2M1M0 = 000 时, 未用(默认); M2M1M0 = 001

时,向器件发送了完整的16位数,且 \overline{CS} 被拉高时,PGA立即进入关断模式;M2M1M0=010时,写至寄存器命令;M2M1M0=011和M2M1M0=1xx时,未用(保留位)。

(2) 增益寄存器。增益寄存器的格式如表7-22所示。

表7-22 增益寄存器的格式(默认值:xxxxx000)

D7	D6	D5	D4	D3	D2	D1	D0
未用	未用	未用	未用	未用	可写位	可写位	可写位
—	—	—	—	—	G2	G1	G0

表7-22中:G2、G1、G0是增益选择位。当G2、G1和G0为000~111时,选择8个增益(1、2、4、5、8、10、16和32)中的一个增益。

(3) 通道寄存器。通道寄存器的格式如表7-23所示。

表7-23 通道寄存器的格式(默认值:xxxxx000)

D7	D6	D5	D4	D3	D2	D1	D0
未用	未用	未用	未用	未用	可写位	可写位	可写位
—	—	—	—	—	C2	C1	C0

表7-23中:C2、C1、C0是增益选择位。当C2、C1和C0为000~111时,选择8个通道中的一个通道(默认000通道)。

3) 关断命令

软件关断命令可使用户将放大器置于低功耗模式(见表7-21)。在这种关断模式下,大多数引脚为高阻抗。PGA一旦进入关断模式,它将一直保持在该模式下,直到向器件发送了一个有效命令(除NOP或关断命令之外)为止,或直到关闭器件电源并重新打开为止。内部寄存器的值在关断时保持不变。一旦退出关断模式,器件将返回到其先前的状态。这样就能使用一个命令使器件退出关断模式,如发送一个命令来选择当前通道(或增益),器件将退出关断模式。

4) 菊花链配置

通过将一个器件的SO引脚连接到下一个器件的SI引脚,并使用公共的SCK和 \overline{CS} 线,可以将多个器件连接为一个菊花链配置。该方式可简化PCB布局。

使用菊花链配置时的工作时序如图7-16所示。在实际使用中可以使用这种方式配置任意数量的器件。MCP6S21和MCP6S22只能用在菊花链的末端,因为它们没有串行数据输出(SO)引脚。SI和SO数据都将以16位(2字节)字为发送单元。这些器件将中止任何字长不是16位的整数倍的命令。

使用菊花链配置时,由于SO引脚的传播时延,可达到的最大时钟频率降为约5.8MHz。每当 \overline{CS} 变为高电平(执行了一个命令)时,内部SPI移位寄存器都会自动装入零。因此,一旦 \overline{CS} 线变为低电平,从SO引脚输出的前16位就总是0。这意味着装入菊花链的下个器件中的第一个命令是NOP。该功能使得在最远的器件无需变化时,可以发送较短的命令和数据字节串。例如,如果链上有三个器件,只有中间的器件需要变化,则只需要发送32个字节的数据(发给第一个和中间的器件),在 \overline{CS} 引脚上升沿执行命令时,链上的最后一个器件将收到NOP。

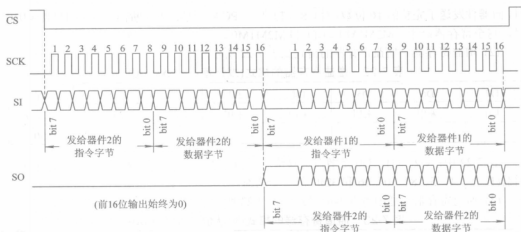


图 7-16 菊花链时序

7.4.2 应用说明

1. 更改外部参考电压

图 7-17(a)给出了 VREF 引脚为 2.5 V，且 $V_{cc} = 5.0\text{ V}$ 时的 MCP6S21 的接法。这种接法使 PGA 可以放大以 2.5 V 为中心的信号，而不是以“地”为参照的电压信号。参考电压 MCP1525 由 MCP6021 缓冲，后者提供了从直流到高频的低输出阻抗的参考电压。驱动 VREF 引脚的电压源的输出阻抗应小于等于 $0.1\ \Omega$ ，以维持合理的增益精度。

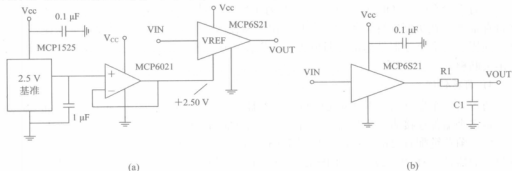


图 7-17 两种放大电路

(a) 具有外部参考电压的 PGA；(b) 容性负载很大时的 PGA

2. 容性负载与稳定性

容性负载很大时，可能会导致稳定性问题，并致使 MCP6S21/2/6/8 系列 PGA 的带宽变窄。这是由于负载电容很大会减小内部放大器的相位裕度和带宽。如果 PGA 需要驱动很大的容性负载，则可使用图 7-17(b)所示的电路。VOUT 处的串联电阻(R1)可在高频时将负载变为阻性，从而改善相位裕度，不过，它不会改善带宽。

$C1 \geq 100\text{ pF}$ 时，R1 的估计值为 $50\ \Omega$ 。该值可在工作台上进行微调。调整 R1 以使阶跃响应过冲和频率响应峰值在所有增益水平上处于可接受范围。

3. 布局问题

良好的 PCB 板布局有助于实现电气特性和典型性能曲线中所示的性能。它也有助于尽可能减少电磁兼容性(Electro-Magnetic Compatibility, EMC)问题。

1) 元件的放置

将功能不同的电路分开放置: 模拟与数字分开, 高速与低速分开, 低功耗与高功耗分开, 因为这将减少串扰。保持敏感走线短而直, 将它们的干扰性元件和走线分隔开来。这对于高频(上升时间短)信号尤其重要。

在距 Vcc 引脚 2.5 mm 范围内, 使用一个 0.1 μF 电源旁路电容器。它必须直接连接到接地层。最好使用多层陶瓷电容器或与之相当的高频电容。

2) 信号耦合

MCP6S21/2/6/8 系列运算放大器的输入引脚为高阻抗。这使得它们特别易受电容耦合噪声的影响。使用接地层有助于缓解该问题。

噪声通过电容耦合时, 可使用接地层提供额外的接地电容; 噪声通过磁耦合时, 使用接地层可减少走线之间的互感。增大走线间的距离会起到很好的效果。

布线时更改其中一条走线的方向也可以减少磁耦合。在受影响的走线旁加装保护走线也能起到一定的作用。保护走线应置于受影响走线的两边, 并与之尽可能靠近。将保护走线的两端接地, 如果走线很长, 则将中间也接地。

3) 高频问题

由于 MCP6S21/2/6/8 PGA 在 $G = 16$ 和 32 时的单位增益可接近 64 MHz, 所以使用良好的 PCB 布局是很重要的。任何高频寄生耦合都可能造成不理想的结果。滤除高频信号(即快速边沿变化率)可有所帮助。要尽可能减少高频问题, 应该:

- (1) 使用完整的接地层和电源层;
- (2) 使用高频表面贴装元件;
- (3) 提供“清洁”的供电电压和旁路;
- (4) 保持走线短而直;
- (5) 尝试使用线性电源。

7.4.3 应用电路与编程

1. 应用电路

1) 增益范围

图 7-18(a)给出了测量电流 I_x 的电路。通过改变 PGA 的增益可提高测量精度。正如手持万用表用不同的量程获得最佳结果一样, 该电路可方便地为小信号设置高增益并为大信号设置低增益。因此 PGA 输出所需的动态范围小于其输入(最多小 30 dB)。

2) 更改 PGA 的增益范围

图 7-18(b)所示的电路是在 MCP6S21 之前使用增益为 +10 的 MCP6021。这将使总体增益范围变为 +10 V/V 到 +320 V/V(从 +1 V/V 到 +32 V/V)。

把增益范围改为较低的增益也很容易, 如图 7-18(c)所示。MCP6021 用作单位增益缓冲, 电阻分压器会将增益范围下调到 +0.1 ~ +3.2 V/V(从 +1 到 +32 V/V)。

3) 扩大 PGA 的增益范围

图 7-18(d)提供了+1~+1024 V/V 的增益范围,比单个 PGA 提供的范围(+1~+32 V/V)大得多。第一个 PGA 提供输入复用功能,而第二个 PGA 只需一个输入。这些器件可以组成菊花链工作方式。

4) 带多个传感器的放大器

多通道 PGA(除 MCP6S21 以外)允许用户选择多个传感器(信号)加到输入端,如图 7-18(e)所示。这些器件也可以更改增益,以分别针对每个传感器进行性能优化。

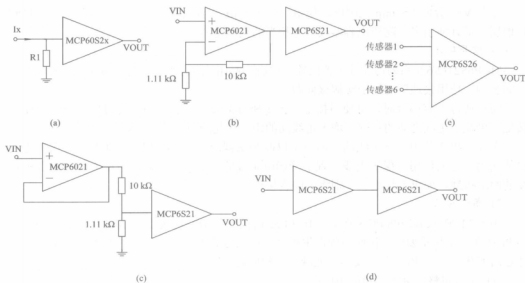


图 7-18 几种应用接法

(a) 电流测量; (b)、(d) 增大增益范围; (c) 减小增益范围; (e) 多路测量

2. 编程方法

对于 MCP6S21/2/6/8, 根据器件原理特别是按照 SPI 接口时序(见图 7-15)及表 7-21~表 7-23 所示的格式就可编写 C51 程序。设单片机的 P1.0、P1.1、P1.2 分别接 MCP6S2x 的 \overline{CS} 、SI 和 SCK。相关源程序代码如下:

```
#include <reg52.h>

sbit CS=P1^0;          /*定义片选 CS 引脚*/
sbit SI=P1^1;          /*定义数据输入脚*/
sbit SCK=P1^2;         /*定义时钟引脚*/

void delay (unsigned char k) /*短延时函数, k 是延时常数*/
{
    unsigned char i;
    for(i=0; i<k; i++);
}

/*MCP6S21/2/6/8 的底层设置函数*/
```

```

void send_MCP6S2x(unsigned char cd,unsigned char numb)
{
    unsigned char i;
    unsigned int j,x;
    SCK=0; delay(5);
    CS=0; delay(5);
    x=(unsigned int)cd*256+numb;    /*将两个字节数据合并成一个整型数据*/
    for(i=0; i<16; i++)            /*送命令+寄存器数据*/
    {
        j=x & 0x8000;
        if(j==0x8000) SI=1;
        else SI=0;
        SCK=1; delay(5);           /*时钟置高*/
        SCK=0; delay(5);           /*时钟置低*/
        x=x<<1;                    /*左移1位*/
    }
    CS=1;                          /*CS 置高*/
}

void main(void)                   /*主函数*/
{
    unsigned char CHX,PGA;
    CS=1; SCK=0;
    CHX=0x02;                      /*设置通道 2*/
    PGA=0x05;                      /*增益为 10*/
    send_MCP6S2x(0x41,CHX);        /*设置模拟通道 2*/
    send_MCP6S2x(0x40,PGA);        /*设置模拟通道 2,增益为 10*/
    send_MCP6S2x(0x20,0x00);       /*进入省电模式*/
    while(1);
}

```

7.5 ISL88705/6/8/13/16 带有看门狗定时器的监控器件

7.5.1 硬件与功能描述

ISL88705/6/8/13/16 系列监控电路能实时监控电源及电池电压和 μP 的工作状态。当电源电压降至其对应门限电压以下时,即产生相应的输出信号。

该系列产品能提供多种功能。每个器件在上电、掉电期间及在电压降低的情况下可产生一个低电平信号。此外,该系列带有一个 1.6 s 的看门狗定时器。复位电平有高电平有效、低电平和开路输出多种形式。具有 1.25 V 门限的电源故障报警电路可用于检测电池电压和

非 5 V 的电源。所有器件都具有手动复位($\overline{\text{MR}}$)输入。看门狗定时器的输出连接到去抖动的
手动复位输入($\overline{\text{MR}}$)将会触发复位信号。

该器件可广泛应用于计算机、控制器、电池供电系统、智能仪表、无线通信系统和手持设备等领域。

1. 主要性能特点

- (1) 固定电压输出, 允许监控器(μP)+3.0 V、+3.3 V 和+5.3 V 供电;
- (2) 供电电源过低输出监控(监视电源);
- (3) 有比较电压(与内部 1.25 V 比较)输入/输出;
- (4) 看门狗超时(超出 1.6 s)复位输出;
- (5) 有高电平复位输出或低电平复位输出, 复位宽度为 160 ms;
- (6) 有手动复位功能;
- (7) 兼容 Maxim 系列或 ADM 系列部分产品;
- (8) 监控电压有(2.65~4.65 V)6 种;
- (9) 电源范围为+2.0~+5.5 V;
- (10) 静态电流<10 μA (在 3 V 供电时);
- (11) 工作温度范围为-40~+85 $^{\circ}\text{C}$ 。

2. 内部结构与引脚说明

ISL88705/6/8/13/16 系列监控电路的内部结构如图 7-19 所示, 引脚排列如图 7-19(b)所示, 外形为 8 脚 DIP、SOP 封装。其中:

(1) $\overline{\text{MR}}$ 是手动复位输入。低电平有效, 内部有 250 μA 的上拉电阻, 允许此脚被 TTL/CMOS 逻辑驱动或由开关短路到地。

(2) Vcc 、 GND 是电源输入端和参考地。

(3) PFI 是电源故障电压监控输入。当 PFI 小于 1.25 V 时, $\overline{\text{PFO}}$ 变为低电平。不用时, PFI 接地或接至 Vcc 。

(4) $\overline{\text{PFO}}$ 是电源故障输出。低电平有效, 且当 PFI 小于 1.25 V 时吸收电流(变低电平)。

(5) WDI 是看门狗(计数器)输入端。 WDI 控制内部看门狗定时器。 WDI 端保持高电平或低电平达 1.6 s 可使内部定时器完成计数, 并将 $\overline{\text{WDO}}$ 变为低电平。将 WDI 悬空或连接到高电平, 或连接到三态缓冲器上将禁止看门狗功能。内部看门狗定时器清零的条件有三种: 发生复位、 WDI 处于三态和 WDI 检测到一个上升沿或下降沿。

(6) $\overline{\text{WDO}}$ 是看门狗输出。当内部看门狗定时器超时 1.6 s 时, $\overline{\text{WDO}}$ 拉至低电平, 并直到看门狗被清零才变为高电平。此外, 当 Vcc 低于复位门限时, $\overline{\text{WDO}}$ 保持低电平。和 RST 不同, $\overline{\text{WDO}}$ 没有最小脉冲宽度, 只要 Vcc 超过复位门限, $\overline{\text{WDO}}$ 就变为高电平而没有延迟。

(7) RST 是低电平有效的复位输出。触发后产生 160 ms 的负脉冲, 并只要 Vcc 低于复位门限, 它就保持低电平, 在 Vcc 上升超过复位门限或 $\overline{\text{MR}}$ 由低电平变为高电平之后, RST 仍保持低电平 160 ms。除非 $\overline{\text{WDO}}$ 连接到 $\overline{\text{MR}}$, 看门狗超时将不会触发 RST 。

(8) RST 是高电平有效的复位输出脚。 RST 与 $\overline{\text{RST}}$ 电平相反。

(9) C_{POF} 是调整复位时间端。通常该引脚外接一个小容值的电容到地。

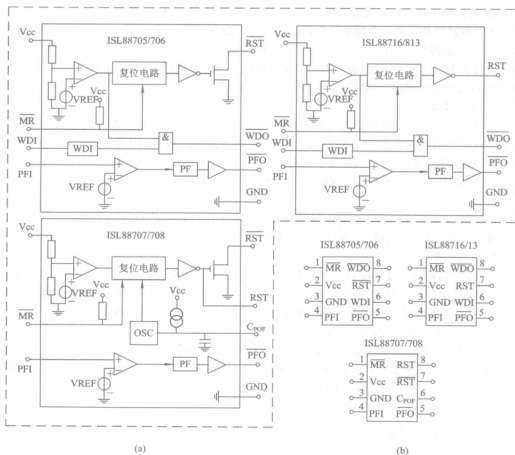


图 7-19 ISL88705/6/8/13/16 系列监控电路的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

3. 使用说明

1) 复位操作

复位信号用来按已知状态启动 MPU(或 CPU), 一旦 MPU 处于未知状态(死机), 就必须使系统复位。

ISL88707/8 有两个复位输出: 一个是高电平有效, 另一个是低电平有效(要外接上拉电阻)。ISL88705/6 只有一个低电平有效(要外接上拉电阻)的复位端。在上电期间, $\overline{\text{RST}}$ 保持低电平直到电源电压升至复位门限(2.65~4.65 V, 与器件种类有关)以上。在超过此门限后, $\overline{\text{RST}}$ 为高电平大约要等 200 ms。

在掉电期间, 当 V_{CC} 降至复位门限以下时, $\overline{\text{RST}}$ 变为低电平, 并在 V_{CC} 大于 1.2 V 时保证低于 0.4 V。

在 V_{CC} 降至复位门限电压以下时, 即处于降压的情况下, $\overline{\text{RST}}$ 变为低电平。如果在已开始的复位脉冲期间电压下降, 则该脉冲至少再持续 140 ms。

2) 用辅助比较器检测电源故障

该系列器件都带有一个辅助比较器,它具有 1.25 V 的转换点和独立的输出端(PFO)及非反相输入端(PFI)。外接一个电阻分压器可使该比较器用作电源电压监控器。PFI 端的衰减电压应设置为仅仅低于 1.25 V 门限。当电源电压下降时, PFI 减小使得 PFO 变为低电平。通常 PFO 接至处理器的中断线上使系统进行相关处理。

3) 手动复位

低电平有效的手动复位输入可被 250 μA 电流上拉到高电平,并可被 CMOS/TTL 逻辑低电平或接地的机械开关驱动至低电位。不需要外部去抖动电路,因为最小为 140 ms 的复位时间可以消除机械按钮开关带来的抖动。

将 WDO 连至 MR 可使看门狗超时产生复位。

4) 看门狗定时器

ISL88xxx 系列内的看门狗定时器监控 $\mu\text{P}/\mu\text{C}$ 的工作。如果在 1.6 s 内未检测到其工作,则内部定时器将使看门狗输出 WDO 处于低电平状态。WDO 将保持低电平直到 WDI 检测到 $\mu\text{P}/\mu\text{C}$ 的工作。

如果将 WDI 悬空或连接到一个三态电路,则看门狗的功能被禁止,即被清零且不计数。如果产生复位信号,则看门狗定时器也会被禁止。当复位信号无效且 WDI 输入检测到短至 50 ns 的高电平或低电平跳变时,看门狗定时器将开始 1.6 s 的计数。WDI 端的跳变会复位看门狗定时器并启动一次新的计数周期。

一旦电源电压 V_{cc} 降至复位门限以下, WDO 也将变为低电平并保持该状态。只要 V_{cc} 升至该门限以上, WDO 就变为高电平。WDO 不存在最小脉冲宽度,因为它是对于复位输出而言的。如果 WDI 悬空,则 WDO 就可作为一个低功耗输出指示器(电源低于复位门限时, WDO 变为低电平)。

7.5.2 应用电路与编程

1. 应用电路

图 7-20 是 ISL88713 用于 51 系列单片机的应用电路。单片机的 P1.4 接至 WDI 作为运行检测信号, WDO 通过二极管接至手动复位脚 MR 上,作为死机后的复位信号。R1 和 R2 的分压比用来检测电源掉电情况(故障的比较输入信号)。当比较后的电源信号小于 1.25 V 时,接于单片机 INT1 的 PFO 变低,作为掉电中断信号。

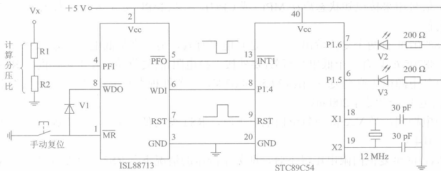


图 7-20 ISL88713 在 51 单片机中的应用

为了正确地使用 ISL88xxx 系列看门狗器件, 应注意以下几点:

(1) 保证 V_{cc} 较低时 \overline{RST} 仍有效。当 V_{cc} 降至 1.1 V 以下时, ISL88705/6/7/8 的 \overline{RST} 输出不再拉低, 它变为不确定。为了避免出现杂散电荷和迫使 \overline{RST} 为低电平, 应该在 \overline{RST} 脚连接一个下拉电阻, 这样可以吸收杂散电荷到地并保持复位信号为低电平。电阻值的要求并不严格, 一个 100 k Ω 的电阻就可将 \overline{RST} 拉至地。

(2) 监控除 V_{cc} 以外的电压。ISL88xxx 可用来检测额外的电源电压。如果将要监控的电压和电源故障输入通过适当的分压比连接到 PFI 脚(与内部+1.25 V 比较), 则只要分压器的电压降至基准电压 1.25 V 以下, \overline{PFO} 输出将变为低电平。如果希望加入迟滞, 则可在 PFI 和 \overline{PFO} 之间接一电阻(其值约是分压器中两电阻之和的 10 倍)。在 PFI 和 GND 之间加一电容可减少电路对输入高频噪声的灵敏度。将 \overline{PFO} 输出连至 \overline{MR} 可以产生一个不同于 \overline{PFO} 标志的复位脉冲或连接到单片机的中断线上。

(3) 监控负电压。电源故障电路也能监视负电源电压。当负电源正常时, \overline{PFO} 为低电平。当负电源变差时, \overline{PFO} 变为高电平(正电压被监控的情况与此相反)。为了触发复位, 这些输出需要加上如图 7-21 所示的电阻和三极管。应该注意: 此电路的精确度取决于 V_{cc} 电源电压、PFI 门限电压的容限以及电阻的精度。

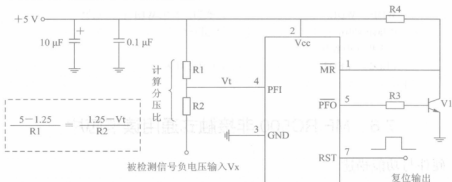


图 7-21 监控负电压

2. 编程方法

系统中加有 ISL88xxx 看门狗器件时, 程序中必须设计有“喂狗”部分, “喂狗”的间隔时间不能超过 1.6 s。另外, 对于监视掉电故障的处理, 一般通过中断实现, 在中断服务程序中, 要根据系统的情况将一些重要数据(或重要事件)及时处理。以图 7-20 为例, 用 C51 实现的源程序如下:

```
#include <reg52.h>
sbit PRO = P3^3;          /*定义 INT1 引脚*/
sbit WDI = P1^7;          /*定义 WDI 输入脚*/
sbit LED1 = P1^6;         /*定义 LED1 灯引脚*/
sbit LED2 = P1^5;         /*定义 LED2 灯引脚*/
void delay(unsigned char k) /*短延时函数, k 是延时常数*/
{
    unsigned char i;
```

```

        for (i=0; i<k; i++){
    }
    /*INT1 中断函数*/
    void INT1_ISL88713( void ) interrupt 2    /*外部中断 1 服务程序*/
    {
        LED1=0;                                /*LED1 点亮*/
        /*掉电, 进入中断, 应根据系统处理关键数据等*/
        /*...*/
        LED1=1;                                /*LED1 灯关*/
    }
    void main(void)                            /*主函数*/
    {
        IT1=1; EX1=1;                        /*开外部中断 1 的下降沿中断*/
        EA=1;                                /*中断允许*/
        LED1=1; LED2=1;                      /*关掉灯*/
        while(1)
        {
            WDI=~WDI;                        /*喂狗, 产生 WDI 电平反转*/
            delay(0x80);                    /*延时*/
            /*主程序内容*/
            LED2=~LED2;                      /*LED2 灯闪烁*/
        }
    }
}

```

7.6 MF RC500 非接触式通用读卡芯片

7.6.1 硬件与功能描述

MF RC500 是应用于非接触式通信中的高集成度读/写卡芯片系列中的一员。该系列读/写卡芯片利用了先进的调制和解调概念, 完全集成了在 13.56 MHz 下所有类型的被动非接触式通信方式和协议, 支持 ISO14443A 的多层应用。该系列芯片内部的发送器部分不需要增加有源电路就能够直接驱动近操作距离的天线(可达 100 mm), 接收器部分提供一个坚固而有效的解调和解码电路, 用于处理 ISO14443A 兼容的应答器信号, 数字部分处理 ISO14443A 帧和错误检测(奇偶&CRC)。此外, MF RC500 采用 CPYPTO1 加密算法, 用于验证 MIFARE 系列产品, 含有非易失性内部密钥存储器, 安全的保密功能让“信息窃贼”无处下手, 方便的并行接口可直接连接到任何 8 位微处理器, 这样给读卡器/终端的设计提供了极大的灵活性。

采用非接触的使用方法避免了频繁接触导致读卡器破坏、金属触点易氧化和污染等传统接触式读/写芯片的弱点, MF RC500 具有复杂的加密算法, 并具有防恶意攻击等安全特性, 已广泛应用于地铁、公交系统、一卡通、小额支付等领域, 成为人们日常生活离不开的“电子钱包”。

1. 主要性能特点

(1) 高集成的解调、解码、模拟电路;

- (2) 输出缓冲功率驱动采用最少数目的外部元件;
- (3) 感应距离可达 100 mm;
- (4) 支持 ISO/IEC14443TypeA 协议的 1~4 部分和 MIFARE 经典协议;
- (5) 采用 CRYPTO1 加密算法并含有安全的非易失性内部密钥存储器;
- (6) 并行微处理器接口带有内部地址锁存和灵活的中断处理;
- (7) 自动识别微处理器并行接口类型和具有 64 字节发送和接收缓冲区;
- (8) 具有低功耗的硬件复位和软件掉电模式;
- (9) 具有可编程定时器和唯一的序列号;
- (10) 用户可编程配置;
- (11) 数字、模拟和发送器部分经独立的引脚分别供电;
- (12) 内部振荡器缓存器连接 13.56 MHz 石英晶振;
- (13) 在短距离应用中, 发送器(天线驱动)可以用 3.3 V 供电。

2. 内部结构与引脚说明

MF RC500 的内部结构与引脚排列如图 7-22 所示。并行微控制器接口自动检测连接的 8 位并行接口的类型。它包含一个易用的双向 FIFO 缓冲区和一个可配置的中断输出。这样就为连接各种 MCU 提供了很大的灵活性。即使使用成本非常低的器件也能满足高速非接触式通信的要求。数据处理部分执行数据的并行-串行转换。它支持的帧包括 CRC 和奇偶校验。它以完全透明的模式进行操作, 因而支持 ISO14443A 的所有层。状态和控制部分允许对器件进行配置以适应环境的影响并使性能调节到最佳状态。

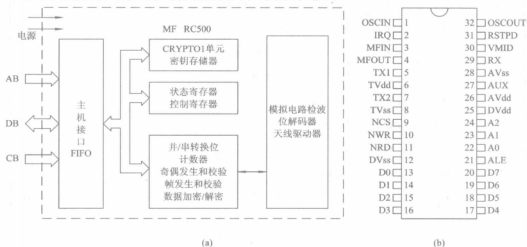


图 7-22 MF RC500 的内部结构与引脚排列

(a) 内部结构; (b) 引脚排列

模拟电路包含了一个具有的阻抗非常低的桥驱动输出部分, 这使得最大操作距离可达 100 mm。接收器可以检测到并解码非常弱的应答信号。

该器件为 32 脚 SO 封装, 见图 7-22(b)。该器件使用了 3 个独立的电源以实现在 EMC 特性和信号去耦方面达到最佳性能。MF RC500 具有出色的 RF 性能并且模拟和数字部分可适应不同的操作电压。其引脚功能见表 7-24。

表 7-24 MF RC500 的引脚功能

管脚	符号	类型	描 述
1	OSCIN	I	晶振振荡器反相输入脚(fosc = 13.56 MHz)
2	IRQ	O	中断请求输出信号
3	MFIN	I	MIFARE 接口输入
4	MFOUT	O	MIFARE 接口输出
5	TX1	O	经过调制的 13.56 MHz 能量载波的发送器 1
6	TVdd	PWR	发送器电源, 提供 TX1 和 TX2 输出电源
7	TX2	O	经过调制的 13.56 MHz 能量载波的发送器 2
8	TVss	PWR	发送器地线, 提供 TX1 和 TX2 的电源参考地
9	NCS	I	选择和激活 MF RC500 的微处理器接口的片选
10	NWR	I	MF RC500 寄存器写入数据 D0~D7 的写选通
11	NRD	I	MF RC500 寄存器读出数据 D0~D7 的读选通
12	DVss	PWR	数字地
13~20	D0~D7	I/O	8 位双向数据总线或地址总线(AD0~AD7)
21	ALE	I	地址锁存信号, 高时将 AD0~AD5 锁存为内部地址
22	A0	I	地址线 0, 寄存器地址位 0
23	A1	I	地址线 1, 寄存器地址位 1
24	A2	I	地址线 2, 寄存器地址位 2
25	DVdd	PWR	数字电源
26	AVdd	PWR	模拟电源
27	AUX	O	辅助输出, 该脚输出模拟测试信号
28	AVss	PWR	模拟地
29	RX	I	接收器输入, 卡应答输入, 该脚经过天线电路耦合调制的 13.56 MHz 载波信号
30	VMID	PWR	内部参考电压, 该脚输出内部参考电压, 必须接一个 100 nF 电容
31	RSTPD	I	当为高时内部灌电流关闭, 振荡器停止, 下降沿启动内部复位
32	OSCOU	O	晶振振荡器输出(反向放大器输出)

3. 并行接口

在每次上电或硬复位后, MF RC500 也复位其并行微处理器接口模式, 并检测当前微处理器接口的类型。检测的原则是根据控制脚的逻辑电平识别微处理器接口。不同微处理器类型的连接电平见表 7-25。

表 7-25 接口连接配置

MF RC500	并行接口类型				
	独立读/写选通		共用读/写选通		
	专用地址总线	复用地址总线	专用地址总线	复用地址总线	握手的复用地址总线
ALE	HIGH	ALE	HIGH	AS	nAStrb
A2	A2	LOW	A2	LOW	HIGH
A1	A1	HIGH	A1	HIGH	HIGH
A0	A0	HIGH	A0	LOW	nWait
NRD	NRD	NRD	NRD	NRD	NDStrb
NWR	NWR	NWR	N/WR	N/WR	NWWrite
NCS	NCS	NCS	NCS	NCS	LOW
D7~D0	D7~D0	AD7~AD0	D7~D0	AD7~AD0	AD7~AD0

4. MF RC500 寄存器

MF RC500 寄存器集合如表 7-26 所示。

表 7-26 MF RC500 寄存器集合

	地址(hex)	寄存器名	功 能
0 页 命 令 和 状 态	0	Page	选择寄存器页
	1	Command	启动(和停止)命令的执行
	2	FIFOData	64 字节 FIFO 缓冲区输入和输出
	3	PrimaryStatus	接收器和发送器以及 FIFO 缓冲区状态标志
	4	FIFOLength	FIFO 中缓冲的字节数
	5	SecondaryStatus	不同的状态标志
	6	InterruptEn	允许和禁止中断请求通过的控制位
1 页 控 制 和 状 态	7	InterruptRq	中断请求标志
	8	Page	选择寄存器页
	9	Control	不同的控制标志,例如定时、节电
	A	ErrorFlag	显示上次命令执行错误状态的错误标志
	B	CollPos	RF 接口检测到的第一个冲突位的位置
	C	TimerValue	定时器的实际值
	D	CRCResultLSB	CRC 处理器寄存器的最低位
2 页 发 送 编 码 控 制	E	CRCResultMSB	CRC 处理器寄存器的最高位
	F	BitFraming	位方式帧的调节
	10	Page	选择寄存器页
	11	TxControl	天线驱动脚 TX1 和 TX2 的逻辑状态控制
	12	CWConductance	选择天线驱动脚 TX1 和 TX2 的电导率
	13	PreSet13	该值不会改变
	14	PreSet14	该值不会改变
	15	ModWidth	选择调整脉冲的宽度
	16	PreSet16	该值不会改变
	17	PreSet17	该值不会改变

续表

	地址(hex)	寄存器名	功 能
3 页 接 收 解 码 控 制	18	Page	选择寄存器页
	19	RxControl1	控制接收器状态
	1A	DecodeControl	控制解码器状态
	1B	BitPhase	选择发送器和接收器时钟之间的位相位
	1C	RxThreshold	选择位解码器的阈值
	1D	PreSet1D	该值不会改变
	1E	RxControl2	控制解码器状态和定义接收器的输入源
	1F	ClockQControl	控制时钟产生用于 90° 相移的 Q 信道时钟
4 页 时 序 信 道 冗 余	20	Page	选择寄存器页
	21	RxWait	选择发送后, 接收器启动前的时间间隔
	22	ChannelRedundancy	选择 RF 信道上数据完整性检测的类型和模式
	23	CRCPreSetLSB	CRC 寄存器预设值的低字节
	24	CRCPreSetMSB	CRC 寄存器预设值的高字节
	25	PreSet25	该值不会改变
	26	MFOUTSelect	选择输出到管脚 MFOUT 的内部信号
	27	PreSet27	该值不会改变
5 页 定 时 和 中 断	28	Page	选择寄存器页
	29	FIFOLevel	定义 FIFO 上溢和下溢警告界限
	2A	TimerClock	选择定时器的时钟的分频器
	2B	TimerControl	选择定时器的起始和停止条件
	2C	TimerReload	定义定时器的预装值
	2D	IRQPinConfig	配置 IRQ 脚的输出状态
	2E	PreSet2E	该值不会改变
	2F	PreSet2F	该值不会改变
6 页地址: 30~37 保留			
7 页 测 试 控 制	38	Page	选择寄存器页
	39	RFU	保留
	3A	TestAnaSelect	选择模拟测试模式
	3B	PreSet3B	该值不会改变
	3C	PreSet3C	该值不会改变
	3D	TestDigiSelect	选择数字测试模式
	3E	RFU	保留
	3F	RFU	保留

1) Page 寄存器的描述

Page 寄存器的地址分别是: 0x00、0x08、0x10、0x18、0x20、0x28、0x30、0x38。

复位值是: 1000000B(0x80)。其位的选择见表 7-27。

表 7-27 Page 位的选择

D7	D6	D5	D4	D3	D2	D1	D0
UsePageSelect	0	0	0	0	PageSelect		

其中:

(1) UsePageSelect(位 7): 如果设置为 1, 则 PageSelect 的值作为寄存器地址 A5、A4 和 A3。寄存器地址的最低位由地址脚或内部地址锁存单独定义。如果设置为 0, 则内部地址所处的整个内容定义寄存器的地址。

(2) 仅当 UsePageSelect(位 2、位 1 和位 0)设置为 1 时, 才使用 PageSelect 的值, 此情况下它指定寄存器页(寄存器地址 A5、A4 和 A3)。

2) Command 寄存器的描述

Command 寄存器的地址是 0x01, 复位值是 0x00。其位的选择见表 7-28。

表 7-28 Command 位的选择

D7	D6	D5	D4	D3	D2	D1	D0
IfDetectBusy	0	Command					

其中:

(1) IfDetectBusy(位 7): 显示接口检测逻辑的状态。设置为 0 表示接口检测成功完成, 设置为 1 表示接口检测正在进行。

(2) Command(位 5~位 0)根据命令代码激活命令。读该寄存器显示实际执行的命令。

3) SecondaryStatus 寄存器的描述

SecondaryStatus 寄存器地址是 0x05, 复位值是 0x60。其位的选择见表 7-29。

表 7-29 SecondaryStatus 位的选择

D7	D6	D5	D4	D3	D2	D1	D0
TRunning	E2Ready	CRCReady	0	0	RxLastBits		

其中:

(1) TRunning(位 7): 如果为 1, 则 MFR500 的定时器单元正在运行, 例如计数器会在下一个定时时钟将 TimerValue 寄存器值减 1。

(2) E2Ready(位 6): 如果为 1, 则 MFR500 已经完成对 E²PROM 的编程。

(3) CRCReady(位 5): 如果为 1, 则 MFR500 已经完成 CRC 的计算。

(4) RxLastBits(位 2~位 0): 显示最后接收字节的有效位数。如果为 0, 则整个字节有效。

4) TimerClock 寄存器的描述

TimerClock 寄存器地址是 0x2A, 复位值是 0x07。其位的选择见表 7-30。

表 7-30 TimerClock 位的选择

D7	D6	D5	D4	D3	D2	D1	D0
0	0	TAutoRestart	TPreScaler				

其中:

(1) TAutoRestart(位 5): 如果为 1, 则定时器从 TReloadValue 处自动重新开始向下计数,

而不是向下计数到零。如果设置为 0, 则定时器减少到零并且 TimerIRq 置位。

(2) TPreScaler(位 4~位 0): 定义定时器时钟。

5) TimerControl 寄存器的描述

TimerControl 寄存器地址是 0x2B, 复位值是 0x06。其位的选择见表 7-31。

表 7-31 TimerControl 位的选择

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	TStopRxEnd	TStopRxBegin	TStartTxEnd	TStartTxBegin

其中:

(1) TStopRxEnd(位 3): 如果为 1, 则当数据接收结束时定时器自动停止。0 表示定时器不受该条件影响。

(2) TStopRxBegin(位 2): 如果为 1, 则当接收到第一个有效位时定时器自动停止。0 表示定时器不受该条件影响。

(3) TStartTxEnd(位 1): 如果为 1, 则当数据发送结束时定时器自动停止。0 表示定时器不受该条件影响。

(4) TStartTxBegin(位 0): 如果为 1, 则当第一个字节发送时定时器自动停止。0 表示定时器不受该条件影响。

6) IRQPinConfig 寄存器的描述

IRQPinConfig 寄存器地址是 0x2D, 复位值是 0x02。其位的选择见表 7-32。

表 7-32 IRQPinConfig 位的选择

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	IRQInv	IRQPushPull

其中:

(1) IRQInv(位 1): 如果为 1, 则管脚 IRQ 上的信号与对应的位 IRQ 状态相反; 如果为 0, 则表示相同。

(2) IRQPushPull(位 0): 如果为 1, 则管脚 IRQ 为标准 CMOS 输出; 如果为 0, 则 IRQ 为开漏输出。

5. 寄存器寻址

可通过如下 3 种机制对 MF RC500 进行操作:

(1) 通过执行命令初始化功能和控制数据寻址;

(2) 通过一系列的配置和功能状态寻址;

(3) 通过读取状态标志监控 MF RC500 的状态寻址。

专用的地址总线和复用的地址总线格式如下所述。

1) 专用的地址总线

使用 MF RC500 专用地址总线, 微处理器通过地址脚 A0、A1 和 A2 定义 3 条地址线。这允许在一页内进行寻址, 即分页模式。专用地址总线的分页模式见表 7-33。

表 7-33 专用地址总线的分页模式

寄存器位	寄存器地址					
1	PageSelect2	PageSelect1	PageSelect0	A2	A1	A0

2) 复用的地址总线

使用 MF RC500 复用的地址总线, 微处理器可以一次定义所有的 6 条地址线。这种情况下既可以使用分页寄存器寻址, 也可使用线性寻址。其地址组合情况见表 7-34。

表 7-34 复用地址总线的地址组合情况

总线类型	寄存器位	寄存器地址					
分页模式	1	PageSelect2	PageSelect1	PageSelect0	AD2	AD1	AD0
线性寻址	0	AD5	AD4	AD3	AD2	AD1	AD0

6. E²PROM 存储器结构

MF RC500 有 32 个块, 块地址从 0 到 2F, 每块 16 个字节。其中, 第 1 块是产品信息区, 从第 2 块到第 3 块是启动寄存器初始化文件区, 从第 4 块到第 8 块是寄存器初始化文件区, 从第 9 块到第 32 块是 CRYPTO1 密钥区。

7. 中断请求系统

MF RC500 通过在 PrimaryStatus 寄存器中设置中断 IRQ, 并使 IRQ 脚有效, IRQ 脚上的信号可用于具有中断处理能力的微处理器来响应中断。

中断标志 TimerIRq 指示由定时器单元产生的中断。TimerIRq 置位的条件是定时器减到 0 或者 TPreLoad 值(如果 TAutoRestart 使能)。

TxIRq 位指示由不同中断源产生的中断。如果发送器有效而且状态从发送数据到帧结束, 则发送器单元自动将中断标志位置位。CRC 协处理器在处理完 FIFO 缓冲区内所有数据后将 TxIRq 置位, 这通过标志 CRCReady = 1 指示。如果 E²PROM 编程结束, 则 TxIRq 置位, 通过位 E2Ready = 1 指示。

当接收数据的结束被检测到时, 通过 RxIRq 标志指示中断。如果命令结束并且命令寄存器的内容变为 Idle, 则标志 IdleIRq 置位。如果 HiAlert 位置 0, 则标志 HiAlertIRq 置位, 这表示 FIFO 缓冲区已到达由位 WaterLevel 指示的边界。

8. 串行数据转换

MF RC500 包含两个主要模块: 一个数字电路(包括状态机编码器和解码器逻辑等)和一个模拟电路(包括调制器、天线驱动器、接收器和放大电路)。这两个模块的接口可配置为这样一种方式, 即接口信号可输出到管脚 MFIN 和 MFOUT。

这样的拓扑结构支持将 MF RC500 的模拟部分与其他器件的数字部分连接。

9. 重要信号描述

1) 天线

非接触式天线使用了 TX1、TX2(输出缓冲)、WMID(模拟参考电压)和 RX(模拟天线输入信号)。

为了驱动天线, MF RC500 通过 TX1 和 TX2 提供 13.56MHz 的能量载波。根据寄存器的设定对发送数据进行调制可得到发送的信号。

射频卡采用 RF 场的负载调制进行响应。天线拾取的信号经过天线匹配电路送到 RX 脚。MF RC500 内部接收器对信号进行检测和解调并根据寄存器的设定进行处理, 然后将数据

发送到并行接口由微控制器进行读取。

另外, MF RC500 对驱动部分使用单独电源供电, 即 TVdd(发送器电源电压)和 TGND(发送器电源地)。

2) 模拟电源和数字电源

为了实现最佳性能, MF RC500 的模拟部分使用单独电源。它对振荡器、模拟解调器和解码器电路供电。模拟电源为 AVdd(模拟部分电源电压)和 AGND(模拟部分电源地)。数字部分也采用单独供电, 即 DVdd(数字部分电源电压)和 DGND(数字部分电源地)。

3) 振荡器

13.56 MHz 晶振通过快速片内缓冲区连接到 OSCIN 和 OSCOUT。如果器件采用外部时钟, 则可从 OSCIN 输入。

4) MIFARE 接口

MF RC500 支持 MIFARE 有源天线的概念。它可以处理管脚 MFIN 和 MFOUT 处的 MIFARE 核心模块的基带信号 NPAUSE 和 KOMP。

MIFARE 接口可采用下列方式与 MF RC500 的模拟或数字部分单独通信。

(1) 模拟电路可通过 MIFARE 接口独立使用。在这种情况下, MFIN 连接到外部产生的 NPAUSE 信号, MFOUT 提供 KOMP 信号。

(2) 数字电路可通过 MIFARE 接口驱动外部信号电路。在这种情况下, MFOUT 提供内部产生的 NPAUSE 信号, 而 MFIN 连接到外部输入的 KOMP 信号。

7.6.2 典型应用电路

MF RC500 支持不同的微控制器接口。一个智能的自动检测逻辑可以自动适应系统总线的并行接口。使用信号 NCS 选择芯片。要使用独立的地址和数据总线与微控制器相连, 必须将 ALE 脚连接到 DVdd。若使用复用的地址和数据总线与微控制器接口, 则必须将 ALE 脚连接到微控制器的 ALE 信号。

若要使用 RNW 和 NDS(取代 NWR 和 NRD)与微控制器相连, 则微控制器的 RNW 必须连接到管脚 NWR, 而 NDS 必须连接到 NRD。

1. 典型接法

MF RC500 与单片机的典型连接如图 7-23 所示。该电路直接配以合适的天线即可使用。匹配电路包括一个 EMC 低通滤波器、接收电路、天线匹配电路和天线。

2. 电路描述

1) EMC 低通滤波器

MIFARE 系统在 13.56 MHz 频率下工作。该频率由一个石英晶振产生用于驱动 MF RC500 以及作为驱动天线的 13.56 MHz 能量载波的基频。这样不仅会产生 13.56 MHz 的发射功率, 而且会发射更高的谐波。国际 EMC 条例定义了广播频段中发射功率的幅值, 因此要符合这一规范必须对输出信号进行适当的滤波。

对于多层电路板, 一定要使用图 7-23 所示的低通滤波器。低通滤波器包括元件 L0 和 C0, 表 7-35 给出了它们的值。

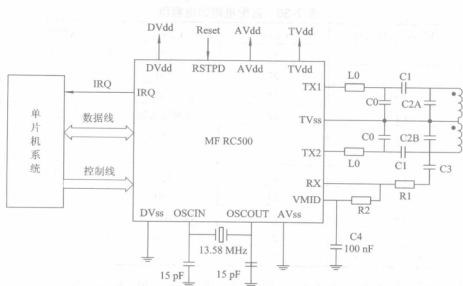


图 7-23 MF RC500 与单片机的典型连接电路

2) 接收电路

MF RC500 的内部接收部分使用受益于副载波的双边带技术。推荐使用内部产生的 VMID 电势作为 RX 脚的输入电势。为了提供一个稳定的参考电压，必须在 VMID 脚接一个对地的电容 C4。读卡器的接收部分需要在 RX 和 VMID 脚之间连接一个分压器。此外，在天线线圈和分压器之间推荐使用一个串接的电容。图 7-23 所示的电路包括了推荐的接收电路。接收电路包括的元件有 R1、R2、C3 和 C4，它们的值见表 7-35。

表 7-35 EMC 滤波器和接收电路的元件参数

元 件	参 考 值	备 注
L0	$2.2 \mu\text{H} \pm 10\%$	磁屏蔽器件，例如 TDK ACL3225S-T
C0	$47 \text{ pF} \pm 2\%$	NPO 材料，根据天线的电感量取值
R1	$10 \text{ k}\Omega \pm 5\%$	金属膜电阻
R2	$820 \Omega \pm 5\%$	金属膜电阻
C3	$15 \text{ pF} \pm 2\%$	NPO 材料

3) 直接匹配天线的阻抗

推荐将天线设计成环形或者矩形，且要设计一个直接匹配天线的匹配电路(建议使用图 7-23 所示的电路)。电容 C1、C2A 和 C2B 的值取决于天线的电气特性和环境的影响，见表 7-36。

表 7-36 匹配电路的电容值

天线线圈电感量/ μH	C1/pF	C2A/pF	C2B/pF
0.8	27	270	330
0.9	27	270	270
1.0	27	220	270
1.1	27	180 22(并联)	220
1.2	27	180	180 22(并联)
1.3	27	180	180
1.4	27	150	180
1.5	27	150	150
1.6	27	120 10(并联)	150
1.7	27	120	150
1.8	27	120	120

然而,要得到最佳的性能必须通过调谐过程找到精确的值。假设天线线圈的杂散电容值为 15 pF。电容 C1 和 C2 应当采用允差为 2% 的 NPO 电介质。

天线的电感和电容的实际值取决于不同的参数,例如:

- (1) 天线的结构(PCB 的类型);
- (2) 导体的厚度;
- (3) 绕组之间的距离;
- (4) 屏蔽层;
- (5) 附近环境中的金属或铁氧体等。

3. 与 MIFARE CLASSIC 有关的内容

1) CRYPTO1 卡验证

为了对 MIFARE CLASSIC 产品进行正确的验证,采用了快速的 CRYPTO1 流密码。对应的密钥必须编程到 MF RC500 的非易失性密钥存储器中。

软件只需要发送两条命令即可打开处于 CRYPTO1 保护下的通信。

2) 启动卡的验证

必须从内部密钥存储器中选择用于卡验证的正确密钥,并将其装入内部 CRYPTO1 寄存器。接下来将验证命令发送到卡。当从卡接收到第一个报文令牌后,微控制器必须检测通信状态标志。如果通信到目前为止保持成功,则启动卡验证的第二部分。

3) 卡验证的第二部分

在这一阶段中,发送到卡的数据由 MF RC500 内部的 CRYPTO1 单元自动产生。要执行这部分动作必须启动相应的命令,卡将会以第二个报文令牌进行响应。然后微控制器必须检测通信状态标志。如果验证已成功,则与 MIFARE CLASSIC 卡继续进行 CRYPTO1 加密下的通信。

4. MF RCxxx 系列器件选型

MF RCxxx 系列器件选型指南见表 7-37。

表 7-37 MF RCxxx 器件器件选型指南

性能(参数)	器件型号				
	MF RC500	MF RC530	MF RC531	MF RC523	CL RC5632
最大工作距离/mm	100	100	100	80	100
FIFO/B	64	64	64	64	64
主机接口	8 位并行	8 位并行、SPI	8 位并行、SPI	SPI、I ² C、UART	8 位并行、SPI
载波频率/MHz	13.56	13.56	13.56	13.56	13.56
模拟接口	完全兼容	完全兼容	完全兼容	完全兼容	完全兼容
调制方式	ASK	ASK	ASK	ASK	ASK
波特率/(kb/s)	106	106/212/424	106/212/424	106/212/424	106/212/424
ISO14443A	Y	Y	Y	Y	Y
ISO14443B	—	—	Y	Y	Y
MIFARE 经典协议	Y	Y	Y	Y	Y
DVdd/V	5.0	3.3/5/0	2.5~3.6	2.5~3.6	3.3/5/0
AVdd/V	5.0	5.0	5.0	5.0	5.0
静态电流/ μ A	2	2	2	1	2
唤醒时间/ μ s	1000	1000	1000	1000	1000
工作温度/ $^{\circ}$ C	-25~+85	-25~+85	-25~+85	-25~+85	-25~+85
封装	SO-32	SO-32	SO-32	HVQFN-32	SO-32

7.6.3 编程方法

采用图 7-23 的典型连接, 单片机选用 STC89C54RD, 相关连接信息如表 7-38 所示。

表 7-38 MF RC500 与单片机的连接关系信息

STC89C54RD(晶振频率 11.0592 MHz)		MF RC500	
引脚	信号名	引脚	信号名
1	P1.0(复位信号)	31	RSTPD
2	P1.1(片选信号)	9	NCS
36	P3.6($\overline{\text{WR}}$)写信号	10	NWR
37	P3.7($\overline{\text{RD}}$)读信号	11	NRD
30	ALE(锁存信号)	21	ALE
39	P0.0(D0) 数据总线	13	D0
38	P0.1(D1) 数据总线	14	D1
37	P0.2(D2) 数据总线	15	D2
36	P0.3(D3) 数据总线	16	D3
35	P0.4(D4) 数据总线	17	D4
34	P0.5(D5) 数据总线	18	D5

续表

STC89C54RD(晶振频率 11.0592 MHz)		MF RC500	
引脚	信号名	引脚	信号名
33	P0.6(D6) 数据总线	19	D6
32	P0.7(D7) 数据总线	20	D7
12	P3.2(INT0)中断	2	IRQ
21	P2.0(地址)	22	A0
22	P2.1(地址)	23	A1
23	P2.2(地址)	24	A2

相关 C51 程序如下:

```

#include <reg52.h>
#include <intrins.h>
#include <stdio.h>
#include <absacc.h>

sbit RST = P1^0;      /*定义复位引脚*/
sbit IRQ = P3^2;      /*定义外部中断 1 引脚*/
sbit CS = P1^1;       /*定义片选信号引脚*/

#define GetRegPage(addr) (0x80 | (addr>>3))

/*****常用操作定义*****/
#define BYTE unsigned char
#define WORD unsigned int
#define uint16 unsigned int

/*****RC500 命令宏定义*****/

#define RegPage 0x00
#define RegCommand 0x01
#define RegFIFOData 0x02
#define RegPrimaryStatus 0x03
#define RegFIFOLength 0x04
#define RegSecondaryStatus 0x05
#define RegInterruptEn 0x06
#define RegInterruptRq 0x07
#define RegControl 0x09
#define RegErrorFlag 0x0A
#define RegCollPos 0x0B
#define RegTimerValue 0x0C
#define RegCRCResultLSB 0x0D
#define RegCRCResultMSB 0x0E
#define RegBitFraming 0x0F

```

#define	RegTxControl	0x11
#define	RegCwConductance	0x12
#define	RegCoderControl	0x14
#define	RegModWidth	0x15
#define	RegRxControl1	0x19
#define	RegDecoderControl	0x1A
#define	RegBitPhase	0x1B
#define	RegRxThreshold	0x1C
#define	RegRxControl2	0x1E
#define	RegClockQControl	0x1F
#define	RegRxWait	0x21
#define	RegChannelRedundancy	0x22
#define	RegCRCPresetLSB	0x23
#define	RegCRCPresetMSB	0x24
#define	RegMfOutSelect	0x26
#define	RegFIFOLevel	0x29
#define	RegTimerClock	0x2A
#define	RegTimerControl	0x2B
#define	RegTimerReload	0x2C
#define	RegIrqPinConfig	0x2D
#define	RegTestAnaSelect	0x3A
#define	RegTestDigiSelect	0x3D
#define	RegTestDigiAccess	0x3F
#define	PCD_IDLE	0x00
#define	PCD_WRITEE2	0x01
#define	PCD_READE2	0x03
#define	PCD_LOADCONFIG	0x07
#define	PCD_LOADKEYE2	0x0B
#define	PCD_AUTHENT1	0x0C
#define	PCD_CALCCRC	0x12
#define	PCD_AUTHENT2	0x14
#define	PCD_RECEIVE	0x16
#define	PCD_LOADKEY	0x19
#define	PCD_TRANSMIT	0x1A
#define	PCD_TRANSCEIVE	0x1E
#define	PCD_RESETPHASE	0x3F
#define	PICC_REQIDL	0x26
#define	PICC_REQALL	0x52
#define	PICC_ANTICOLL1	0x93

```

#define PICC_ANTICOLL2 0x95
#define PICC_ANTICOLL3 0x97
#define PICC_AUTHENT1A 0x60
#define PICC_AUTHENT1B 0x61
#define PICC_READ 0x30
#define PICC_WRITE 0xA0
#define PICC_DECREMENT 0xC0
#define PICC_INCREMENT 0xC1
#define PICC_RESTORE 0xC2
#define PICC_TRANSFER 0xB0
#define PICC_HALT 0x50

/*****数据传送结构体宏定义*****/
#define MI_REQUESTERR 10 /*寻卡错误*/
#define MI_ANTICOLLERR 11 /*防冲突错误*/
#define MI_SELECTERR 12 /*选卡错误*/
#define MI_REQUEST 20 /*寻卡成功*/
#define MI_ANTICOLL 21 /*防冲突成功*/
#define MI_SELECT 22 /*选卡成功*/

typedef struct
{
    BYTE cmd;
    BYTE status;
    BYTE nBytesSent;
    BYTE nBytesToSend;
    BYTE nBytesReceived;
    WORD nBitsReceived;
    BYTE CardNumber[5];
    BYTE SerBuffer[20];
}MfCmdInfo;

#define ResetInfo(Info)
    Info.cmd = 0; \
    Info.status = 0; \
    Info.nBytesSent = 0; \
    Info.nBytesToSend = 0; \
    Info.nBytesReceived = 0; \
    Info.nBitsReceived = 0; \

MfCmdInfo Info; /*变量定义*/
void Delay_us(uint16 n) /*延时程序*/
{
    uint16 i;
    for(i=0; i<n; i++) _nop_();
}

```

```

void UART_Init(void)          /*串口初始化*/
{
    SCON = 0x50;
    TMOD |= 0x20;
    PCON |= 0x80;
    TH1 = 0xff;               /*57600 b/s 对 11.0592 MHz*/
    TL1 = 0xff;
    TR1 = 1;
}

/*-----由串口发单个字符-----*/
void sendchar(unsigned char ch)
{
    SBUF = ch;
    while(TI == 0);
    TI = 0;    Delay_us(10);    /*延时*/
}

void WriteRawIO(unsigned char Address, unsigned char value)
{
    XBYTE[Address] = value;
}

BYTE ReadRawIO(unsigned char Address)
{
    return XBYTE[Address];
}

void WriteIO(unsigned char Address, unsigned char value)    /*硬件驱动程序*/
{
    WriteRawIO(0x00, GetRegPage(Address));
    WriteRawIO(Address, value);
}

BYTE ReadIO(unsigned char Address)
{
    WriteRawIO(0x00, GetRegPage(Address));
    return ReadRawIO(Address);
}

BYTE SetBitMask(unsigned char reg, unsigned char mask)    /*置一位*/
{
    BYTE tmp = 0x00;
    tmp = ReadIO(reg);
    WriteIO(reg, tmp | mask);    /*set bit mask*/
    return 0x00;
}

BYTE ClearBitMask(unsigned char reg, unsigned char mask)    /*清一位*/
{
    BYTE tmp = 0x00;
    tmp = ReadIO(reg);
    WriteIO(reg, tmp & ~mask);    /*clear bit mask*/
    return 0x00;
}

```

```

BYTE M500PcdRfReset(unsigned char val)
{
    if(val)
    {
        ClearBitMask(RegTxControl,0x03);
        Delay_us(2000);
        SetBitMask(RegTxControl,0x03);
    }
    else ClearBitMask(RegTxControl,0x03);
    return 0;
}

BYTE M500PcdReset(void) /*RC500 功能程序*/
{
    uint16 timecnt;
    RST=0; Delay_us(100);
    RST=1; Delay_us(100);
    RST=0; Delay_us(100);
    timecnt=1000;
    while( ( ReadIO(RegCommand) & 0x3F ) & timecnt--);
    if(!timecnt)return 1;
    if(ReadIO(RegCommand)!=0x00)return 1;
    return 0;
}

BYTE M500PcdConfig(void) /*配置 RC500 内部寄存器函数*/
{
    if (!M500PcdReset())
    {
        WriteIO(RegClockQControl,0x00); /*Q 时钟复位*/
        WriteIO(RegClockQControl,0x40); /*Q 时钟写入 0x40*/
        Delay_us(150); /*延时 100 μs 以上*/
        WriteIO(RegClockQControl,0x00); /*Q 时钟清零*/
        WriteIO(RegBitPhase,0xAD); /*写 BitPhase 寄存器*/
        WriteIO(RegRxThreshold,0xFF); /*RxThreshold 寄存器写入 0xFF*/
        WriteIO(RegRxControl2,0x01); /*RxControl2 寄存器写入 0x01*/
        WriteIO(RegFIFOLevel,0x1A); /*设置 FIFO 上溢和下溢的界限*/
        WriteIO(RegTimerControl,0x02); /*设置数据发送完成后停止定时器*/
        WriteIO(RegIrqPinConfig,0x03); /*配置 IRQ 引脚的输出状态*/
        M500PcdRfReset(1); /*打开天线*/
        WriteIO(RegMfOutSelect,0x02); /*选择输出到 MFOUT 为来自内部编码的调制信号*/
        return 0;
    }
    else return 1;
}

BYTE M500PcdCmd(void) /*发送命令*/
{
    BYTE i,n;

```

```

BYTE lastBits, irqEn = 0x00, waitFor = 0x00;
WORD timecnt = 0;
switch(Info.cmd)
{
    case PCD_IDLE:
        irqEn = 0x00; waitFor = 0x00; break;
    case PCD_WRITEE2:
        irqEn = 0x11; waitFor = 0x10; break;
    case PCD_READE2:
        irqEn = 0x07; waitFor = 0x04; break;
    case PCD_LOADCONFIG:
    case PCD_LOADKEYE2:
    case PCD_AUTHENT1:
        irqEn = 0x05; waitFor = 0x04; break;
    case PCD_CALCCRC:
        irqEn = 0x11; waitFor = 0x10; break;
    case PCD_AUTHENT2:
        irqEn = 0x04; waitFor = 0x04; break;
    case PCD_RECEIVE:
        irqEn = 0x06; waitFor = 0x04; break;
    case PCD_LOADKEY:
        irqEn = 0x05; waitFor = 0x04; break;
    case PCD_TRANSMIT:
        irqEn = 0x05; waitFor = 0x04; break;
    case PCD_TRANSCEIVE:
        irqEn = 0x3D; waitFor = 0x04; break;
    default: return 1;
}
WriteIO(RegInterruptEn, 0x7F); /*清除中断使能*/
WriteIO(RegInterruptRq, 0x7F); /*清除中断标志*/
WriteIO(RegCommand, PCD_IDLE); /*清除 Command 寄存器*/
SetBitMask(RegControl, 0x01); /*清除 FIFO 指针*/
for(j=0; i<Info.nBytesToSend; i++)
    WriteIO(RegFIFOData, Info.SerBuffer[i]); /*将数据写入 FIFO*/
irqEn |= 0x20; waitFor |= 0x20;
WriteIO(RegInterruptEn, irqEn | 0x80); /*打开中断*/
WriteIO(RegCommand, Info.cmd); /*将命令写入命令寄存器*/
timecnt=1000;
while(!(ReadIO(RegInterruptRq) & waitFor || !(timecnt--)));
WriteIO(RegInterruptEn, 0x7F); /*清除中断使能*/

```

```

WriteIO(RegInterruptRq,0x7F);      /*清除中断标志寄存器*/
SetBitMask(RegControl,0x04);        /*停止定时器*/
WriteIO(RegCommand,PCD_IDLE);      /*清除命令寄存器*/
if(!timecnt)
{ printf("timeout!\n"); return 1; }  /*超时退出, 返回 1*/
if(ReadIO(RegErrorFlag)&0x17)        /*如果产生错误, 则返回 1*/
{ sendchar(ReadIO(RegErrorFlag)&0x3F); printf("ERROR!\n"); return 1; }
if(Info.cmd == PCD_TRANSCEIVE)
{ n = ReadIO(RegFIFOLength);         /*读取 FIFO 中数据的总字节数*/
  lastBits = ReadIO(RegSecondaryStatus)&0x07;
  Info.nBytesReceived = n;
  if(lastBits) Info.nBitsReceived=(n-1)*8 + lastBits;
  else Info.nBitsReceived = n * 8;
  if(n==0) n=1;
  for(i=0; i<n; i++)
    Info.SerBuffer[i]=ReadIO(RegFIFOData);
}
return 0;
}

BYTE M500PiccCommonRequest(void)    /*寻卡*/
{ WriteIO(RegTimerClock,0x09);       /*设置定时器分频*/
  WriteIO(RegTimerReload,0xA0);       /*设置定时器初值*/
  ClearBitMask(RegControl,0x08);      /*寄存器 Control 清 0*/
  WriteIO(RegBitFraming,0x07);        /*寄存器 BitFramming 写入 07H*/
  WriteIO(RegChannelRedundancy,0x03); /*选择 RF 信道上数据完整性检测的类型和模式*/
  SetBitMask(RegTxControl,0x03);      /*打开天线*/
  Info.cmd = PCD_TRANSCEIVE;          /*命令为发送命令*/
  Info.nBytesToSend = 1;              /*发送字节长度为 1*/
  Info.nBytesReceived = 2;            /*接收字节长度为 2*/
  Info.SerBuffer[0] = PICC_REQIDL;    /*数据为寻卡命令*/
  if(M500PcdCmd()) return 1;          /*调用发送命令子函数*/
  if(Info.nBitsReceived != 16) return 1; /*判断接收到的是否为 16 位, 若不是, 则是错误的*/
  if(Info.SerBuffer[0]==0x04) return 0; /*如果第一个字节是 0x04, 则寻卡成功*/
  else return 1;                      /*否则返回错误状态*/
}

BYTE M500PiccCascAnticoll(void)      /*防冲突*/
{ BYTE i;
  BYTE snr_check=0;
  BYTE status=0;
  WriteIO(RegTimerClock,0x07);        /*设置定时器分频*/

```

```

WriteIO(RegTimerReload,0x6A);          /*设置定时器初值*/
WriteIO(RegDecoderControl,0x28);        /*防冲突处理*/
ClearBitMask(RegControl,0x08);
WriteIO(RegChannelRedundancy,0x03);
Info.cmd = PCD_TRANSCEIVE;              /*命令为发送命令*/
Info.nBytesToSend = 2;                   /*发送字节长度为 2*/
Info.nBytesReceived = 5;                 /*接收字节长度为 5*/
Info.SerBuffer[0] = PICC_ANTICOLL1;      /*数据 0 为防冲突命令*/
Info.SerBuffer[1] = 0x20;
if(M500PcdCmd()) status = 1;
else
{
    for(i=0; i<4; i++)
        snr_check^=Info.SerBuffer[i];    /*生成校验数据*/
    if(snr_check!=Info.SerBuffer[i])      /*对比校验数据是否正确*/
        status = 1;
    else
    {
        for(i=0; i<5; i++)
            Info.CardNumber[i] = Info.SerBuffer[i]; /*将卡号和 CRC 装到缓存*/
    }
}
ClearBitMask(RegDecoderControl,0x20);    /*结束防冲突处理*/
return status;
}

BYTE M500PiccCascSelect(void)            /*选卡*/
{
    BYTE i;
    WriteIO(RegTimerClock,0x07);          /*设置定时器分频*/
    WriteIO(RegTimerReload,0x6A);         /*设置定时器初值*/
    WriteIO(RegChannelRedundancy,0x0F);
    ClearBitMask(RegControl,0x08);
    Info.cmd = PCD_TRANSCEIVE;
    Info.nBytesToSend = 7;                /*发送字节长度为 7*/
    Info.nBytesReceived = 1;              /*接收字节长度为 1*/
    Info.SerBuffer[0] = PICC_ANTICOLL1;   /*数据 0 为选卡命令*/
    Info.SerBuffer[1] = 0x70;
    for(i=0; i<5; i++)
        Info.SerBuffer[i+2] = Info.CardNumber[i];
    if(M500PcdCmd()) return 1;
    if(Info.SerBuffer[0] != 0x08) return 1;
    return 0;
}

```



```

void M500HostCodeKey(BYTE *uncoded,BYTE *coded) /*密码格式转换*/
{
    BYTE cnt = 0,ln = 0,hn = 0;
    for (cnt=0; cnt<6; cnt++)
    {
        ln = uncoded[cnt] & 0x0F;
        hn = uncoded[cnt] >> 4;
        coded[cnt * 2 + 1] = (~ln << 4) | ln;
        coded[cnt * 2] = (~hn << 4) | hn;
    }
}

BYTE M500PiccAuthState(BYTE Block) /*Cryptol 卡验证*/
{
    BYTE i;
    Info.cmd = PCD_AUTHENT1;
    Info.nBytesToSend = 0x06; /*发送字节长度为 6*/
    Info.nBytesReceived = 0; /*接收字节长度为 0*/
    Info.SerBuffer[0] = PICC_AUTHENT1A; /*AUTHEN1 命令*/
    Info.SerBuffer[1] = Block; /*块号*/
    for(i=0; i<4; i++) /*卡号*/
        Info.SerBuffer[i+2] = Info.CardNumber[i];
    if(M500PcdCmd()) return 1; /*调用发送命令子函数*/
    if(ReadIO(RegSecondaryStatus)&0x07) return 1;
    Info.cmd = PCD_AUTHENT2; /*AUTHEN2 命令*/
    Info.nBytesToSend = 0; /*发送数据为 0 字节*/
    Info.nBytesReceived = 0; /*接收数据为 0 字节*/
    if(M500PcdCmd()) return 1; /*调用发送命令子函数*/
    if(ReadIO(RegControl)==0x08) return 0; /*值是否为 0x08*/
    else return 1;
}

BYTE M500PiccAuthKey(BYTE *KEY,BYTE Sector,BYTE Block) /*直接验证密码*/
{
    Info.cmd = PCD_LOADKEY;
    Info.nBytesToSend = 0x0C; /*发送字节长度为 12*/
    Info.nBytesReceived = 0; /*接收字节长度为 0*/
    M500HostCodeKey(KEY,Info.SerBuffer); /*将密码转换成固定格式*/
    if(M500PcdCmd()) return 1; /*调用发送命令子函数*/
    if(M500PiccAuthState(Sector*4+Block)) return 1; /*CRYPTOL 卡验证子程序*/
    return 0;
}

BYTE M500PiccRead(BYTE Sector,BYTE Block) /*读数据*/
{
    BYTE tmp = 0;
    WriteIO(RegTimerClock,0x09); /*设置定时器分频*/
}

```

```

    WriteIO(RegTimerReload,0xA0);          /*设置定时器初值*/
    WriteIO(RegChannelRedundancy,0x0F);
    ResetInfo(Info);
    Info.cmd = PCD_TRANSCIVE;
    Info.SerBuffer[0] = PICC_READ;
    Info.SerBuffer[1] = Sector*4+Block;
    Info.nBytesToSend = 2;
    Info.nBytesReceived = 0x10;
    if(M500PcdCmd()) return 1;              /*调用发送命令子函数*/
    if(Info.nBytesReceived != 16) return 1;
    for(tmp=0; tmp<0x10; tmp++)
        sendchar(Info.SerBuffer[tmp]);
    return 0;
}

BYTE M500PiccWrite(BYTE Sector,BYTE Block) /*写数据*/
{
    Info.cmd = PCD_TRANSCIVE;
    Info.SerBuffer[0] = PICC_WRITE;
    Info.SerBuffer[1] = Sector*4+Block;
    Info.nBytesToSend = 2;
    Info.nBytesReceived = 0;
    if(M500PcdCmd()) return 1;              /*调用发送命令子函数*/
    WriteIO(RegTimerClock,0x09);            /*设置定时器分频*/
    WriteIO(RegTimerReload,0xA0);           /*设置定时器初值*/
    WriteIO(RegChannelRedundancy,0x0F);
    Info.cmd = PCD_TRANSCIVE;
    Info.nBytesToSend = 0x10;
    Info.nBytesReceived = 0;
    if(M500PcdCmd()) return 1;              /*调用发送命令子函数*/
    if(Info.SerBuffer[0]==0x0A) return 0;
    return 1;
}

BYTE M500PcdLoadKeyE2(WORD StartAddr) /*装载密码到EEPROM*/
{
    BYTE key[6];
    Info.cmd = PCD_LOADKEYE2;
    Info.nBytesToSend = 2;                  /*发送字节长度为2*/
    Info.SerBuffer[0] = StartAddr & 0xFF;  /*数据0为防冲突命令*/
    Info.SerBuffer[1] = StartAddr >> 8;
    M500HostCodeKey(key,&Info.SerBuffer[2]);
    if(M500PcdCmd()) return 1;
    return 0;
}

```

